

ALAGAPPA UNIVERSITY

(Accredited with 'A+' Grade by NAAC (CGPA: 3.64) in the Third Cycle
and Graded as Category –I University by MHRD-UGC)
(A State University Established by the Government of Tamilnadu)



KARAIKUDI – 630 003

DIRECTORATE OF DISTANCE EDUCATION

M.Sc. (Computer Science)

III-SEMESTER

341 43

ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS

SYLLABI-BOOK MAPPING TABLE

ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS

Syllabi	Mapping in Book
BLOCK 4: AI IN ROBOTICS	
UNIT X State Space Search Block word & robot example Path selection Monkey & Banana Problem AND – OR Graph Means end Analysis in a robotic problem Robot problem solving as a production system Triangle table Robot learning	**Pages 279-336**
UNIT XI Robot task planning Phases in task planning Symbolic spatial relationships Obstacle avoidance Graph planning	**Pages 337-352**
BLOCK 5: MACHINE VISION	
UNIT XII Introduction Functions in a vision system Imaging devices Lighting A-D Conversion Quantization Encoding Imaging Storage Image data reduction	**Pages 353-397**
UNIT XIII Segmentation Techniques Feature Extraction Object Recognitions	**Pages 398-427**
UNIT XIV Training the Vision System Robotic applications of machine vision	**Pages 428-457**

UNIT I - PROBLEMS AND SEARCH

Structure

1.1 Introduction

"The science and engineering of making intelligent machines, especially intelligent computer programs". -John McCarthy-

Artificial Intelligence is an approach to make a computer, a robot, or a product to think how smart human think. AI is a study of how human brain think, learn, decide and work, when it tries to solve problems. And finally, this study outputs intelligent software systems. The aim of AI is to improve computer functions which are related to human knowledge, for example, reasoning, learning, and problem-solving.

The intelligence is intangible. It is composed of

- Reasoning
- Learning
- Problem Solving

- Perception
- Linguistic Intelligence

The objectives of AI research are reasoning, knowledge representation, planning, learning, natural language processing, realization, and ability to move and manipulate objects. There are long-term goals in the general intelligence sector.

Intelligence, as we know, is the ability to acquire and apply the knowledge. Knowledge is the information acquired through experience. Experience is the knowledge gained through exposure(training). Summing the terms up, we get **artificial intelligence** as the "copy of something natural(i.e., human beings) 'WHO' is capable of acquiring and applying the information it has gained through exposure."

Intelligence is composed of:

- Reasoning
- Learning
- Problem Solving
- Perception
- Linguistic Intelligence

Many tools are used in AI, including versions of search and mathematical optimization, logic, methods based on probability and economics. The AI field draws upon computer science, mathematics, psychology, linguistics, philosophy, neuro-science, artificial psychology and many others.

Artificial Intelligence is composed of two words **Artificial** and **Intelligence**, where Artificial defines *"man-made,"* and intelligence defines *"thinking power"*, hence AI means *"a man-made thinking power."*

So, we can define AI as: "It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like humans, and able to make decisions."

Artificial Intelligence exists when a machine can have human based skills such as learning, reasoning, and solving problems

With Artificial Intelligence you do not need to preprogram a machine to do some work, despite that you can create a machine with programmed algorithms which can work with own intelligence, and that is the awesomeness of AI.



It is believed that AI is not a new technology, and some people says that as per Greek myth, there were Mechanical men in early days which can work and behave like humans.

Why Artificial Intelligence?

Before Learning about Artificial Intelligence, we should know that what is the importance of AI and why should we learn it. Following are some main reasons to learn about AI:

- With the help of AI, you can create such software or devices which can solve real-world problems very easily and with accuracy such as health issues, marketing, traffic issues, etc.
- With the help of AI, you can create your personal virtual Assistant, such as Cortana, Google Assistant, Siri, etc.
- With the help of AI, you can build such Robots which can work in an environment where survival of humans can be at risk.
- AI opens a path for other new technologies, new devices, and new Opportunities.

Goals of Artificial Intelligence

Following are the main goals of Artificial Intelligence:

1. Replicate human intelligence
2. Solve Knowledge-intensive tasks
3. An intelligent connection of perception and action
4. Building a machine which can perform tasks that requires human intelligence such as:
    - Proving a theorem
    - Playing chess
    - Plan some surgical operation
    - Driving a car in traffic

5. Creating some system which can exhibit intelligent behavior, learn new things by itself, demonstrate, explain, and can advise to its user.

What Comprises to Artificial Intelligence?

Artificial Intelligence is not just a part of computer science even it's so vast and requires lots of other factors which can contribute to it. To create the AI first we should know that how intelligence is composed, so the Intelligence is an intangible part of our brain which is a combination of **Reasoning, learning, problem-solving perception, language understanding, etc**.

To achieve the above factors for a machine or software Artificial Intelligence requires the following discipline:

- Mathematics
- Biology
- Psychology
- Sociology
- Computer Science
- Neurons Study
- Statistics



Advantages of Artificial Intelligence

Following are some main advantages of Artificial Intelligence:

- **High Accuracy with less errors:** AI machines or systems are prone to less errors and high accuracy as it takes decisions as per pre-experience or information.

- **High-Speed:** AI systems can be of very high-speed and fast-decision making, because of that AI systems can beat a chess champion in the Chess game.
- **High reliability:** AI machines are highly reliable and can perform the same action multiple times with high accuracy.
- **Useful for risky areas:** AI machines can be helpful in situations such as defusing a bomb, exploring the ocean floor, where to employ a human can be risky.
- **Digital Assistant:** AI can be very useful to provide digital assistant to the users such as AI technology is currently used by various E-commerce websites to show the products as per customer requirement.
- **Useful as a public utility:** AI can be very useful for public utilities such as a self-driving car which can make our journey safer and hassle-free, facial recognition for security purpose, Natural language processing to communicate with the human in human-language, etc.

1.2 Concepts of AI

1. Recommendation engine

Everyone uses YouTube and Google regularly. To understand these apps and similar search engines better, you must know that they are powered by AI as well. The recommendations that you get when you type something is called predictive searching. This system is made using lots of data about the user collected by Google and may include data like location, age and other personal data. Using this data, the AI predicts what you might be interested in and suggests accordingly. Not only Google and Facebook, but most services like Amazon, Flipkart, Spotify and others use it.

2. Autonomous

Autonomy simply means the machines doing all the work by themselves — with negligible help from humans. Many companies use autonomous AI for a variety of things to reduce human labour. For example, robots are programmed to pick things up and put it at a particular space. A more complex and advanced example of this would be the use of AI for autonomous cars, something which has active research going on worldwide. Some of the Indian efforts in this regard include Fisheybox, ATImotors and Auro Robotics. Autonomous has an AI in its backbone and is an everyday encountered technology, which is why everyone must be aware of it.

3. NLP

Natural Language Processing (NLP) is the ability of a machine to be able to learn human languages — spoken and written. Because we use language to interact with our devices, NLP became an integral part of our lives. Every non-technical person should know NLP because it is being deployed at a number of places like chatbots, the speech-to-text technologies and AI assistants like Siri, Alexa and Cortana. Autocorrect in mobile phones is another example of NLP.

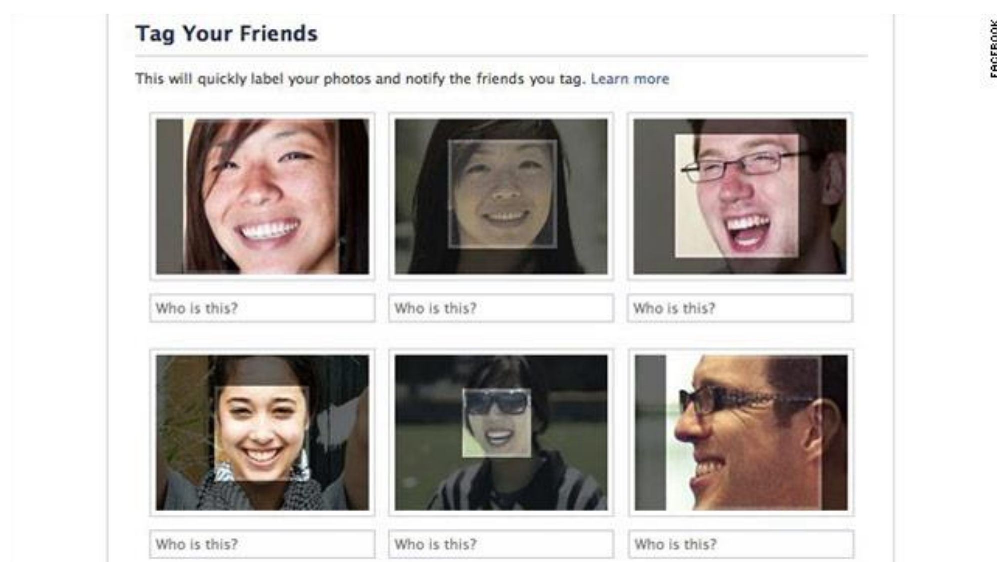
4. Turing Test

With the advent of the terror revolving around AI, the Turing test is a method that checks whether a system can be called an AI or not. The Turing Test determines whether a computer can think like a human or not, testing the qualification of it as an AI. According to an interrogator, if he is unable to distinguish between a computer and a human, the AI passes the Turing Test.

5. Facial Recognition:

Everyone must know about this technology backed by AI because of the influence that it has in various systems around. The system works by noting down various nodal points of the face and then saves the resulting data as a faceprint. Facial detection is used for fraud detection and in some airports at the security checks to make sure if the identity of the person matches correctly.

Mobile phone cameras use facial recognition to unlock phones. Facebook uses the same technology to help you tag your friends on pictures that you upload. All of this is the work of AI.



6. Spam Filters

A spam message to your colleague, might not be a spam message to you. Every person has a varied preference. Today, on Gmail, we have intelligence spam filters that help you get only the messages that you wish to receive with the help of AI. The system works by marking what the user chooses to mark as spam and not spam and learning from the choice of the user of what he may think is spam. The AI technology used for this is the artificial neural network. This system learns individual preferences of spam emails by learning from their previous experiences.

7. Smart Assistants

The assistants like Alexa, Siri, Cortana and even Google Assistant can perform a number of tasks like making appointments for you, setting reminders for you, playing music for you on demand. All these technologies commonly used are backed by AI. These smart assistants used on smartphones and households learn from instances and use NLP to perform all the desired tasks.

8. Language Translators

One of the most difficult challenges that AI faces is language processing. In many cases, language requires not just words, but the contextual information around it as well, in order to entirely understand it. Text to speech applications is an example of this. Other applications include analysing the resource workload and helping with optimising the resource pool of languages, helping to pick the best language for a particular job and so on.

9. Voice To Text

The technology to recognise spoken works and be able to convert into text is voice to text. Google Assistant on our phones does that while browsing on the search engine. Many voice recognition products are becoming popular in the market and people use them on a daily basis. This is why everyone should know about it and the fact that it is supported by AI.

1.3 Approaches of AI

An algorithm is a kind of container. It provides a box for storing a method to solve a particular kind of a problem. Algorithms process data through a series of well-defined states. The states need not be deterministic, but the states are defined nonetheless. The goal is to create an output that solves a problem. In some cases, the algorithm receives inputs that help define the output, but the focus is always on the output.

Algorithms must express the transitions between states using a well-defined and formal language that the computer can understand. In processing the data and solving the problem, the algorithm defines, refines, and executes a function. The function is always specific to the kind of problem being addressed by the algorithm.

Each of the five tribes has a different technique and strategy for solving problems that result in unique algorithms. Combining these algorithms should lead eventually to the master algorithm that will be able to solve any given problem. The following discussion provides an overview of the five main algorithmic techniques.

Symbolic reasoning

One of the earliest tribes, the symbolists, believed that knowledge could be obtained by operating on symbols (signs that stand for a certain meaning or event) and deriving rules from them. By putting together complex systems of rules, you could attain a logic deduction of the result you wanted to know, thus the symbolists shaped their algorithms to produce rules from data. In symbolic reasoning, *deduction* expands the realm of human knowledge, while *induction* raises the level of human knowledge. Induction commonly opens new fields of exploration, while deduction explores those fields.

Connections modelled on the brain's neurons

The connectionists are perhaps the most famous of the five tribes. This tribe strives to reproduce the brain's functions by using silicon instead of neurons. Essentially, each of the neurons (created as an algorithm that models the real-world counterpart) solves a small piece of the problem, and using many neurons in parallel solves the problem as a whole.

The use of backpropagation, or backward propagation of errors, seeks to determine the conditions under which errors are removed from networks built to resemble the human neurons by changing the *weights* (how much a particular input figures into the result) and *biases* (which features are selected) of the network. The goal is to continue changing the weights and biases until such time as the actual output matches the target output. At this point, the artificial neuron fires and passes its solution along to the next neuron in line. The solution created by just one neuron is only part of the whole solution. Each neuron passes information to the next neuron in line until the group of neurons creates a final output. Such a method proved the most effective in human-like tasks such as recognizing objects, understanding written and spoken language, and chatting with humans.

Evolutionary algorithms that test variation

The evolutionaries rely on the principles of evolution to solve problems. In other words, this strategy is based on the survival of the fittest (removing any solutions that don't match the desired output). A fitness function determines the viability of each function in solving a problem. Using a tree structure, the solution method looks for the best solution based on function output. The winner of each level of evolution gets to build the next-level functions. The idea is that the next level will get closer to solving the problem but may not solve it completely, which means that another level is needed. This particular tribe relies heavily on recursion and languages that strongly support recursion to solve problems. An interesting output of this strategy has been algorithms that evolve: One generation of algorithms actually builds the next generation.

Bayesian inference

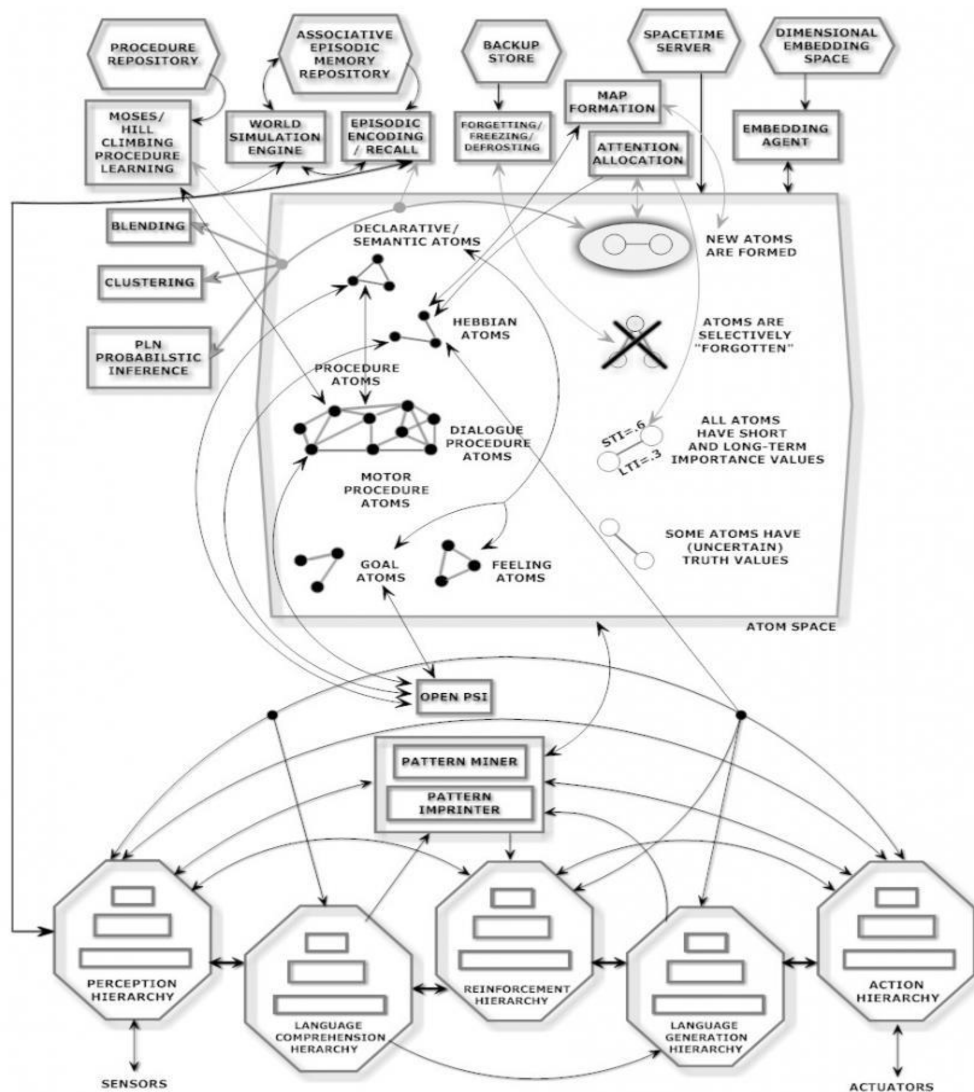
A group of scientists, called Bayesians, perceived that uncertainty was the key aspect to keep an eye on and that learning wasn't assured but rather took place as a continuous updating of previous beliefs that grew more and more accurate. This perception led the Bayesians to adopt statistical methods and, in particular, derivations from Bayes' theorem, which helps you calculate probabilities under specific conditions (for instance, seeing a card of a certain *seed*, the starting value for a pseudo-random sequence, drawn from a deck after three other cards of same seed).

Systems that learn by analogy

The analogyzers use kernel machines to recognize patterns in data. By recognizing the pattern of one set of inputs and comparing it to the pattern of a known output, you can create a problem solution. The goal is to use similarity to determine the best solution to a problem.

It’s the kind of reasoning that determines that using a particular solution worked in a given circumstance at some previous time; therefore, using that solution for a similar set of circumstances should also work. One of the most recognizable outputs from this tribe is recommender systems. For example, when you buy a product on Amazon, the recommender system comes up with other, related products that you might also want to buy.

The ultimate goal of machine learning is to combine the technologies and strategies embraced by the five tribes to create a single algorithm (the master algorithm) that can learn anything. Of course, achieving that goal is a long way off. Even so, scientists such as [Pedro Domingos] are currently working toward that goal.



Neuroscience

The neural approaches inspired by neuroscience try to faithfully reproduce the human brain. My previous blog post Neuromorphic vs. Neural Net explains the differences between neuromorphic approaches with artificial neural networks.

Artificial Neural Network

Artificial Neural Networks, while originally inspired by the human brain in the 1950s, now continue to refer to that basic original architecture and no longer have an explicit goal of accurately modeling the human brain.

Most promising in this category are Neural Turing Machines (such as being commercialized by DeepMind, acquired by Google). NTMs add internal state to conventional neural networks.

Machine learning, especially neural networks, have been dominating headlines to the point of eclipsing the rest of AI, such as historic symbolic AI described below. The downside of neural networks is their opacity -- they are "black boxes" and there is no way to infuse human-curated knowledge directly.

Neural/Symbolic Integration

Why settle for either the black box but adaptive approach of neural networks or the white box but less flexible symbolic approaches, when you can have both? It's quite a challenge and there have been a few different projects meeting various levels of success in various domains. LIDA is one example of this genre.

Cybernetics and brain simulation

In the 1940s and 1950s, a number of researchers explored the connection between neurology, information theory, and cybernetics. Some of them built machines that used electronic networks to exhibit rudimentary intelligence, such as W. Grey Walter’s turtles and the Johns Hopkins Beast. Many of these researchers gathered for meetings of the Teleological Society at Princeton University and the Ratio Club in England. By 1960, this approach was largely abandoned. First problem was that building hardware that simulates neurological processes requires a too many components, and it would he physically hard to connect such large number of neurons as human has. Nowadays some scientist are getting also back to this approach.

Sub-symbolic

By the 1980s progress in symbolic AI seemed to stall and many believed that symbolic systems would never be able to imitate all the processes of human cognition, especially perception, robotics, learning and pattern recognition. A number of researchers began to look into "sub-symbolic" approaches to specific AI problems.

Researchers from the related field of robotics, such as Rodney Brooks, rejected symbolic AI and focused on the basic engineering problems that would allow robots to move and survive. Their work revived the non-symbolic viewpoint of the early cybernetics researchers of the 1950s and reintroduced the use of control theory in AI.

Interest in neural networks and "connectionism" was revived by David Rumelhart and others in the middle 1980s. These and other sub-symbolic approaches, such as fuzzy systems and evolutionary computation, are now studied collectively by the emerging discipline of computational intelligence.

Statistical approach to artificial intelligence

In the 1990s, AI researchers developed sophisticated mathematical tools to solve specific subproblems. These tools are truly scientific, in the sense that their results are both measurable and verifiable, and they have been responsible for many of AI's recent successes. The shared mathematical language has also permitted a high level of collaboration with more established fields (like mathematics, economics or operations research). Stuart Russell and Peter Norvig describe this movement as nothing less than a "revolution" and "the victory of the neats." Critics argue that these techniques are too focused on particular problems and have failed to address the long term goal of general intelligence.

Integrating the approaches

An intelligent agent is a system that perceives its environment and takes actions which maximize its chances of success. The simplest intelligent agents are programs that solve specific problems. More complicated agents include human beings and organizations of human beings (such as firms). The paradigm gives researchers license to study isolated problems and find solutions that are both verifiable and useful, without agreeing on one single approach. An agent that solves a specific problem can use any approach that works – some agents are symbolic and logical, some are sub-symbolic neural networks and others may use new approaches.

1.4 Applications of AI

Real World Artificial Intelligence Applications

Just the mention of AI and the brain invokes pictures of Terminator machines destroying the world. Thankfully, the present picture is significantly more positive. So, let's explore how AI is helping our planet and at last benefiting humankind. In this blog on Artificial Intelligence applications, I'll be discussing how AI has impacted various fields like marketing, finance, banking and so on.

If you're new to AI make sure to check out this blog on *what is AI*.

The various domains which I'll be covering in this blog are:

1. AI In Marketing
2. AI In Banking
3. AI In Finance
4. AI In Agriculture
5. AI In HealthCare
6. AI In Gaming
7. AI In Space Exploration
8. AI In Autonomous Vehicles
9. AI In Chatbots
10. AI In Artificial Creativity

Artificial Intelligence Applications: Marketing

Marketing is a way to sugar coat your products to attract more customers. We, humans, are pretty good at sugar coating, but what if an algorithm or a bot is built solely for the purpose of marketing a brand or a company? It would do a pretty awesome job!

In the early 2000s, if we searched an online store to find a product without knowing it's exact name, it would become a nightmare to find the product. But now when we search for an item on any e-commerce store, we get all possible results related to the item. It's like these search engines read our minds! In a matter of seconds, we get a list of all relevant items. An example of this is finding the right movies on Netflix.

One reason why we're all obsessed with Netflix and chill is because, Netflix provides highly accurate predictive technology based on customer's reactions to films. It examines millions of records to suggest shows and films that you might like based on your previous actions and choices of films. As the data set grows, this technology is getting smarter and smarter every day.



Artificial Intelligence Applications – AI in Marketing

With the growing advancement in AI, in the near future, it may be possible for consumers on the web to buy products by snapping a photo of it. Companies like CamFind and their competitors are experimenting this already.

Artificial Intelligence Applications: Banking

AI in banking is growing faster than you thought! A lot of banks have already adopted AI-based systems to provide customer support, detect anomalies and credit card frauds. An example of this is HDFC Bank.

HDFC Bank has developed an AI-based chatbot called ***EVA*** (Electronic Virtual Assistant), built by Bengaluru-based Senseforth AI Research.

Since its launch, Eva has addressed over 3 million customer queries, interacted with over half a million unique users, and held over a million conversations. Eva can collect knowledge from thousands of sources and provide simple answers in less than 0.4 seconds.

The use of AI for fraud prevention is not a new concept. In fact, AI solutions can be used to enhance security across a number of business sectors, including retail and finance.

By tracing card usage and endpoint access, security specialists are more effectively preventing fraud. Organizations rely on AI to trace those steps by analyzing the behaviors of transactions.



Artificial Intelligence Applications – AI in Banking

Companies such as MasterCard and RBS WorldPay have relied on AI and ***Deep Learning*** to detect fraudulent transaction patterns and prevent card fraud for years now. This has saved millions of dollars.

Artificial Intelligence Applications: Finance

Ventures have been relying on computers and data scientists to determine future patterns in the market. Trading mainly depends on the ability to predict the future accurately.

Machines are great at this because they can crunch a huge amount of data in a short span. Machines can also learn to observe patterns in past data and predict how these patterns might repeat in the future.

In the age of ultra-high-frequency trading, financial organizations are turning to AI to improve their stock trading performance and boost profit.

One such organization is Japan’s leading brokerage house, Nomura Securities. The company has been reluctantly pursuing one goal, i.e. to analyze the insights of experienced stock traders with the help of computers. After years of research, Nomura is set to introduce a new stock trading system.



Artificial Intelligence Applications – AI in Finance

The new system stores a vast amount of price and trading data in its computer. By tapping into this reservoir of information, it will make assessments, for example, it may determine that current market conditions are similar to the conditions two weeks ago and predict how share prices will be changing a few minutes down the line. This will help to take better trading decisions based on the predicted market prices.

Artificial Intelligence Applications: Agriculture

Here's an alarming fact, the world will need to produce 50 percent more food by 2050 because we're literally eating up everything! The only way this can be possible is if we use our resources more carefully. With that being said, AI can help farmers get more from the land while using resources more sustainably.

Issues such as climate change, population growth, and food security concerns have pushed the industry into seeking more innovative approaches to improve crop yield.

Organizations are using automation and robotics to help farmers find more efficient ways to protect their crops from weeds.



Artificial Intelligence Applications – AI in Agriculture

Blue River Technology has developed a robot called See & Spray which uses computer vision technologies like ***object detection*** to monitor and precisely spray weedicide on cotton plants. Precision spraying can help prevent herbicide resistance.

Apart from this, Berlin-based agricultural tech start-up called PEAT, has developed an application called Plantix that identifies potential defects and nutrient deficiencies in the soil through images.

The image recognition app identifies possible defects through images captured by the user's smartphone camera. Users are then provided with soil restoration techniques, tips, and other possible solutions. The company claims that its software can achieve pattern detection with an estimated accuracy of up to 95%.

Artificial Intelligence Applications: Health Care

When it comes to saving our lives, a lot of organizations and medical care centers are relying on AI. There are many examples of how AI in healthcare has helped patients all over the world.

An organization called Cambio Health Care developed a clinical decision support system for stroke prevention that can give the physician a warning when there's a patient at risk of having a heart stroke.

Another such example is Coala life which is a company that has a digitalized device that can find cardiac diseases.



Artificial Intelligence Applications – AI in Health Care

Similarly, Aifloo is developing a system for keeping track of how people are doing in nursing homes, home care, etc. The best thing about AI in healthcare is that you don't even need to develop a new medication. Just by using an existing medication in the right way, you can also save lives.

Artificial Intelligence Applications: Gaming

Over the past few years, Artificial Intelligence has become an integral part of the gaming industry. In fact, one of the biggest accomplishments of AI is in the gaming industry.

DeepMind's AI-based AlphaGo software, which is known for defeating Lee Sedol, the world champion in the game of GO, is considered to be one of the most significant accomplishment in the field of AI.

Shortly after the victory, DeepMind created an advanced version of AlphaGo called **AlphaGo Zero** which defeated the predecessor in an AI-AI face off. Unlike the original AlphaGo, which DeepMind trained over time by using a large amount of data and supervision, the advanced system, AlphaGo Zero taught itself to master the game.

Other examples of Artificial Intelligence in gaming include the First Encounter Assault Recon, popularly known as F.E.A.R, which is a first-person shooter video game.



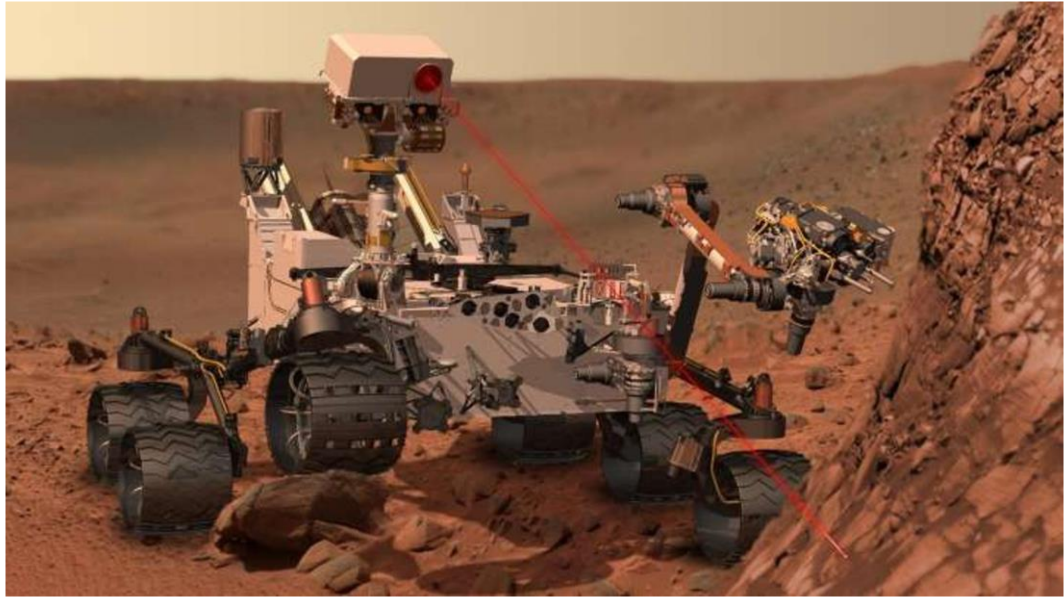
Artificial Intelligence Applications – AI in Gaming

The actions taken by the opponent AI are unpredictable because the game is designed in such a way that the opponents are trained throughout the game and never repeat the same mistakes. They get better as the game gets harder. This makes the game very challenging and prompts the players to constantly switch strategies and never sit in the same position.

Artificial Intelligence Applications: Space Exploration

Space expeditions and discoveries always require analyzing vast amounts of data. Artificial Intelligence and Machine learning is the best way to handle and process data on this scale. After rigorous research, astronomers used Artificial Intelligence to sift through years of data obtained by the Kepler telescope in order to identify a distant eight-planet solar system.

Artificial Intelligence is also being used for NASA's next rover mission to Mars, the Mars 2020 Rover. The AEGIS, which is an AI-based Mars rover is already on the red planet. The rover is responsible for autonomous targeting of cameras in order to perform investigations on Mars.



Artificial Intelligence Applications – Space Exploration

Artificial Intelligence Applications: Autonomous Vehicles

For the longest time, self-driving cars have been a buzzword in the AI industry. The development of autonomous vehicles will definitely revolutionaries the transport system.
Companies like Waymo conducted several test drives in Phoenix before deploying their first AI-based public ride-hailing service. The AI system collects data from the vehicles radar, cameras, GPS, and cloud services to produce control signals that operate the vehicle.

Advanced Deep Learning algorithms can accurately predict what objects in the vehicle's vicinity are likely to do. This makes Waymo cars more effective and safer.

Another famous example of an autonomous vehicle is Tesla's self-driving car. Artificial Intelligence implements computer vision, image detection and deep learning to build cars that can automatically detect objects and drive around without human intervention.

Elon Musk talks a ton about how AI is implemented in tesla's self-driving cars and autopilot features. He quoted that,

"Tesla will have fully self-driving cars ready by the end of the year and a "robotaxi" version – one that can ferry passengers without anyone behind the wheel – ready for the streets next year".



Artificial Intelligence Applications – Autonomous vehicle

Artificial Intelligence Applications: Chatbots

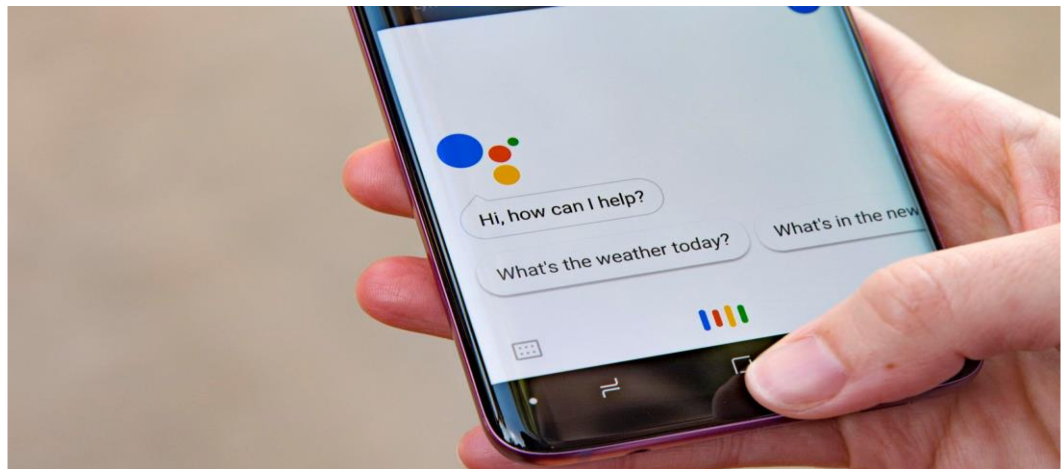
These days Virtual assistants have become a very common technology. Almost every household has a virtual assistant that controls the appliances at home. A few examples include Siri, Cortana, which are gaining popularity because of the user experience they provide.

Amazon’s Echo is an example of how Artificial Intelligence can be used to translate human language into desirable actions. This device uses speech recognition and NLP to perform a wide range of tasks on your command. It can do more than just play your favorite songs. It can be used to control the devices at your house, book cabs, make phone calls, order your favorite food, check the weather conditions and so on.

Another example is the newly released Google’s virtual assistant called Google Duplex, that has astonished millions of people. Not only can it respond to calls and book appointments for you, but it also adds a human touch.



Amazon’s Echo



Google’s Duplex

Artificial Intelligence Applications: Social Media

Ever since social media has become our identity, we’ve been generating an immeasurable amount of data through chats, tweets, posts and so on. And wherever there is an abundance of data, AI and Machine Learning are always involved.

In social media platforms like Facebook, AI is used for face verification wherein machine learning and deep learning concepts are used to detect facial features and tag your friends. Deep Learning is used to extract every minute detail from an image by using a bunch of deep neural networks. On the other hand, Machine learning algorithms are used to design your feed based on your interests.



Artificial Intelligence Application – Social Media

Another such example is Twitter's AI, which is being used to identify hate speech and terroristic language in tweets. It makes use of Machine Learning, Deep Learning, and Natural language processing to filter out offensive content. The company discovered and banned 300,000 terrorist-linked accounts, 95% of which were found by non-human, artificially intelligent machines.

Artificial Intelligence Applications: Artificial Creativity

Have you ever wondered what would happen if an artificially intelligent machine tried to create music and art?

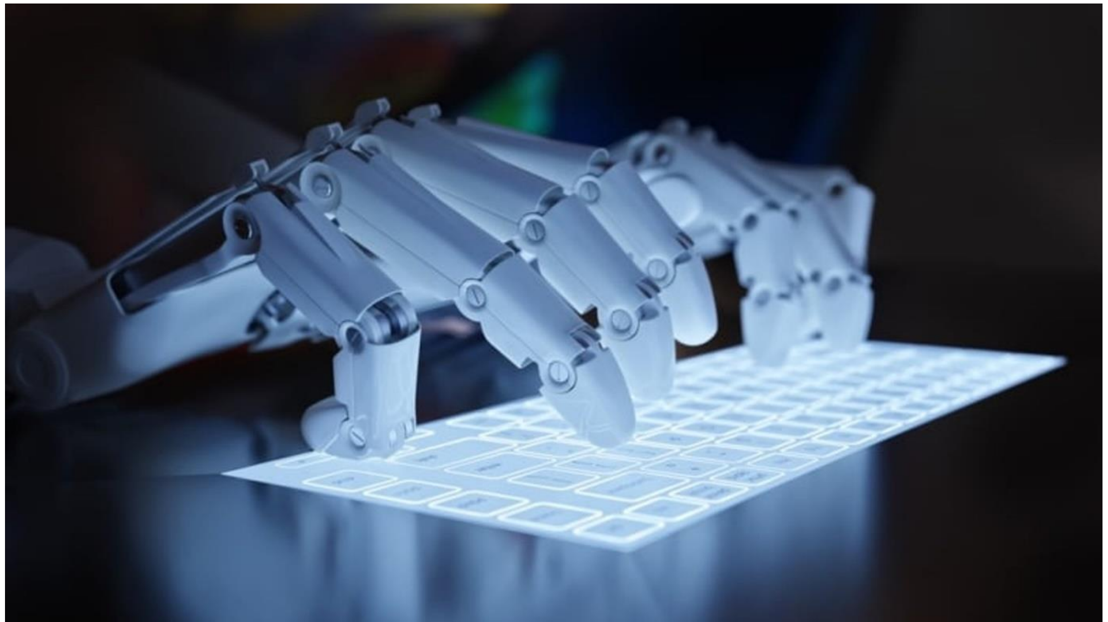
An AI-based system called MuseNet can now compose classical music that echoes the classical legends, Bach and Mozart.

MuseNet is a deep neural network that is capable of generating 4-minute musical compositions with 10 different instruments and can combine styles from country to Mozart to the Beatles.

MuseNet was not explicitly programmed with an understanding of music, but instead discovered patterns of harmony, rhythm, and style by learning on its own.

Another creative product of Artificial Intelligence is a content automation tool called Wordsmith. Wordsmith is a natural language generation platform that can transform your data into insightful narratives.

Tech giants such as Yahoo, Microsoft, Tableau, are using WordSmith to generate around 1.5 billion pieces of content every year.



Artificial Intelligence Application – Artificial Creativity

Check your Progress

Note: a. Write your answer in the space given below

b. Compare your answer with those given at the end of the unit.

i. Define the concepts of AI

…………………………………………………………………………………

…………………………………………………………………………………

1.5 Problem Formulation

Goal-based agents

There are two kinds of goal-based agents: **problem-solving agents** and **planning agents**. Problem-solving agents consider each states of the world as **indivisible**, with no internal structure of the states visible to the problem-solving algorithms. Planning agents split up each state into **variables** and establishes **relationship** between them.

In this series, we will discuss more on problem-solving agents and the algorithms associated with them. We'll keep the discussion on the planning agents for some other time.

In this post (and further too), as an example to explain the various algorithms, we consider the problem of traveling from one place to another (single-source single-destination path). Figure 1 gives the road-map of a part of Romania.

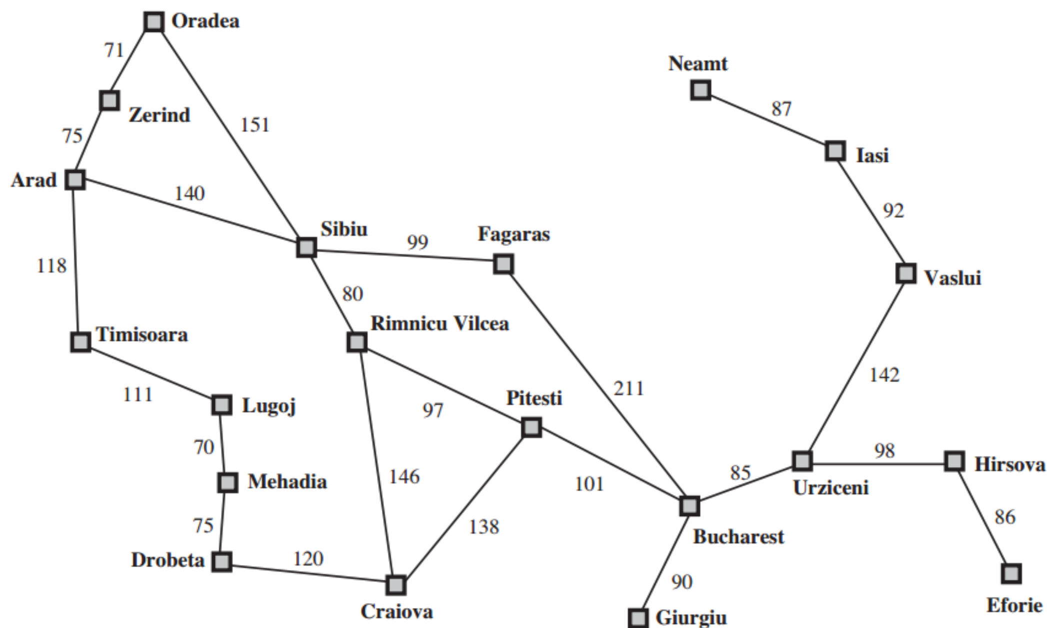


Figure 1: A simplified road map of part of Romania.

The problem is to travel from Arad to Bucharest in a day. For the agent, the goal will be to reach Bucharest the following day. Courses of action that doesn't make agent to reach Bucharest on time can be rejected without further consideration, making the agent's decision problem simplified.

Problem definition and formulation

Before we jump on to finding the algorithm for evaluating the problem and searching for the solution, we first need to define and formulate the problem.

Problem formulation involves deciding what actions and states to consider, given the goal. For example, if the agent were to consider the action to be at the level of "move the left foot by one inch" or "turn the steering wheel by 1 degree left", there would be too many steps for the agent to leave the parking lot, let alone to Bucharest. In general, we need to abstract the state details from the representation.

A problem can be defined formally by 5 components:

1. The **initial state** of the agent. In this case, the initial state can be described as *In: Arad*
2. The possible **actions** available to the agent, corresponding to each of the state the agent resides in. For example, ACTIONS(*In: Arad*) = {*Go: Sibiu*, *Go: Timisoara*, *Go: Zerind*}
3. The **transition model** describing what each action does. Let us represent it by RESULT(s, a) where s is the state the action is currently in and a is the action performed by the agent. In this example, RESULT(*In: Arad, Go: Zerind*) = *In: Zerind.*
4. The **goal test**, determining whether the current state is a goal state. Here, the goal state is {*In: Bucharest*}
5. The **path cost** function, which determine the cost of each path, which is reflecting in the performance measure. For the agent trying to reach Bucharest, time is essential, so we can set the cost function to be the distance between the places. (Here, we are ignoring the other factors that influence the traveling time). By convention, we define the cost function as *c(s, a, s')*, where s is the current state and a is the action performed by the agent to reach state s'.

The initial state, the actions and the transition model together define the **state space** of the problem — the set of all states reachable by any sequence of actions. Figure 1 is the graphical representation of the state space of the traveling problem. A **path** in the state space is a sequence of states connected by a sequence of actions.

The **solution** to the given problem is defined as the sequence of actions from the initial state to the goal states. The quality of the solution is measured by the cost function of the path, and an **optimal solution** has the lowest path cost among all the solutions.

Searching for Solutions

We can form a **search tree** from the state space of the problem to aid us in finding the solution. The initial state forms the **root** node and the **branches** from each node are the possible actions from the current node (state) to the child nodes (next states).

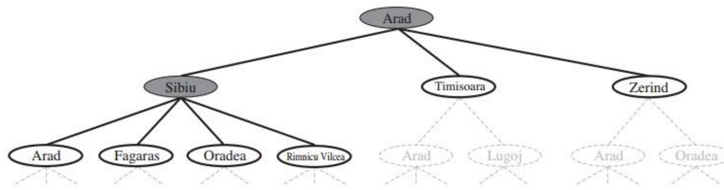


Figure 2: Partial search tree for finding route from Arad to Bucharest. The nodes Arad and Sibiu are opened

The six nodes in Figure 2, which don't have any children (at least until now) are **leaf nodes**. The set of all leaf nodes available for expansion at any given point is called the **frontier**. The search strategy involves the expansion of the nodes in the frontier until the solution (or the goal state) is found (or there are no more nodes to expand).

We have to notice one peculiar thing in the search tree in Figure 2. There is a path from Arad to Sibiu, and back to Arad again. We say that *In(Arad)* is a repeated state, generated by a **loopy path**. This means that the search tree for Romania is *infinite,* even though the search space is limited. These loopy paths makes some of the algorithms to fail, making the problem seem unsolvable. In fact, a loopy path is a special case of **redundant paths**, where there are more than one paths from one state to another (for example, Arad — Sibiu and Arad — Zerind — Oradea — Sibiu).

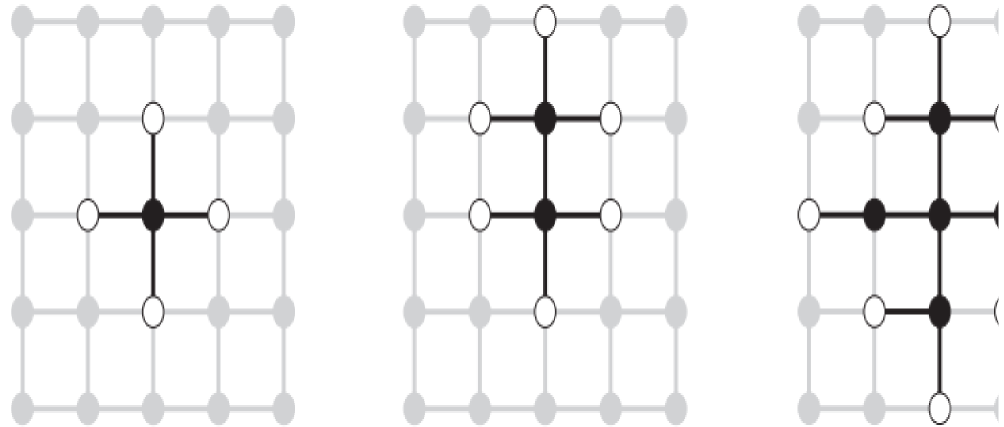
The redundant path situation occurs in almost every problem, and often makes the solution algorithm less efficient, worsening the performance of the searching agent. One way to eliminate the redundancy is to utilize the advantage given by the problem definition itself. For example, in the case of traveling from Arad to Bucharest, since the path costs are additive and step costs are non-negative, only one path among the various redundant paths has the least cost (and it is the shortest distance between the two states), and loopy paths are never better than the same path with loops removed.

Another idea to avoid exploring redundant paths is to remember which states have been visited previously. Along with the search tree, an **explored set** is maintained which contains all the states previously visited. Newly generates which matches the previously generated nodes can be discarded. In this way, every step moves the states in the frontier into the explored region, and some states in the unexplored region into the frontier, until the solution is found.

Performance measure of Problem-solving Algorithms

We can evaluate an algorithm's performance with these metrics:

1. **Completeness**: Is the algorithm guaranteed to find a solution if there exist one?
2. **Optimality**: Does the algorithm find the optimal solution?



3. **Time complexity**: How long does it take for the algorithm to find a solution?
4. **Space complexity**: How much memory is consumed in finding the solution?

In graph theory, the time and space complexity is measured using |V| and |E|, where V and E are the number of vertices and the number of edges in the graph respectively. But in AI, we explore the state space (which is a graph) of the problem using its equivalent search tree. So it is meaningful if we use *b* and *d* to measure the complexity, where *b* is the **branching factor** of the tree (maximum number of successors of any node) and *d* is the **depth** of the shallowest goal node.

1.6 Forward and Backward Reasoning

In Artificial intelligence, the purpose of the search is to find the path through a problem space. There are two ways to pursue such a search that are forward and backward reasoning. The significant difference between both of them is that forward reasoning starts with the initial data towards the goal. Conversely, backward reasoning works in opposite fashion where the purpose is to determine the initial facts and information with the help of the given results.

Definition of Forward Reasoning

The solution of a problem generally includes the initial data and facts in order to arrive at the solution. These unknown facts and information is used to deduce the result. For example, while diagnosing a patient the doctor first check the symptoms and medical condition of the body such as temperature, blood pressure, pulse, eye colour, blood, etcetera.

After that, the patient symptoms are analysed and compared against the predetermined symptoms. Then the doctor is able to provide the medicines according to the symptoms of the patient. So, when a solution employs this manner of reasoning, it is known as **forward reasoning**.

Steps that are followed in the forward reasoning

The inference engine explores the knowledge base with the provided information for constraints whose precedence matches the given current state.

- In the first step, the system is given one or more than one constraints.
- Then the rules are searched in the knowledge base for each constraint. The rules that fulfil the condition are selected(i.e., IF part).
- Now each rule is able to produce new conditions from the conclusion of the invoked one. As a result, THEN part is again included in the existing one.



Basis for comparison	Forward Reasoning	Backward Reasoning
Basic	Data-driven	Goal driven
Begins with	New Data	Uncertain conclusion
Objective is to find the	Conclusion that must follow	Facts to support the conclusions
Type of approach	Opportunistic	Conservative
Flow	Incipient to consequence	Consequence to incipient

Definition of Backward Reasoning

The **backward reasoning** is inverse of forward reasoning in which goal is analysed in order to deduce the rules, initial facts and data. We can understand the concept by the similar example given in the above definition, where the doctor is trying to diagnose the patient with the help of the inceptive data such as symptoms. However, in this case, the patient is experiencing a problem in his body, on the basis of which the doctor is going to prove the symptoms. This kind of reasoning comes under backward reasoning.

Steps that are followed in the backward reasoning

In this type of reasoning, the system chooses a goal state and reasons in the backward direction. Now, let's understand how does it happens and what steps are followed.

- Firstly, the goal state and the rules are selected where the goal state reside in the THEN part as the conclusion.
- From the IF part of the selected rule the subgoals are made to be satisfied for the goal state to be true.
- Set initial conditions important to satisfy all the subgoals.
- Verify whether the provided initial state matches with the established states. If it fulfils the condition then the goal is the solution otherwise other goal state is selected.

Key Differences Between Forward and Backward Reasoning in AI

1. The forward reasoning is data-driven approach while backward reasoning is a goal driven.
2. The process starts with new data and facts in the forward reasoning. Conversely, backward reasoning begins with the results.
3. Forward reasoning aims to determine the result followed by some sequences. On the other hand, backward reasoning emphasis on the acts that support the conclusion.
4. The forward reasoning is an opportunistic approach because it could produce different results. As against, in backward reasoning, a specific goal can only have certain predetermined initial data which makes it restricted.
5. The flow of the forward reasoning is from the antecedent to consequent while backward reasoning works in reverse order in which it starts from conclusion to incipient.

The forward and backward reasoning are differentiated on the basis of their purpose and process, in which forward reasoning is directed by the initial data and intended to find the goal while the backward reasoning is governed by goal instead of the data and aims to discover the basic data and facts.

1.7 Graphs & Trees

Graph Searching

In this chapter, we abstract the general mechanism of searching and present it in terms of searching for paths in directed graphs. To solve a problem, first define the underlying search space and then apply a search algorithm to that search space. Many problem-solving tasks can be transformed into the problem of finding a path in a graph. Searching in graphs provides an appropriate level of abstraction within which to study simple problem solving independent of a particular domain.

A (directed) graph consists of a set of nodes and a set of directed arcs between nodes. The idea is to find a path along these arcs from a start node to a goal node.

The abstraction is necessary because there may be more than one way to represent a problem as a graph. Whereas the examples in this chapter are in terms of state-space searching, where nodes represent states and arcs represent actions, future chapters consider different ways to represent problems as graphs to search.

Formalizing Graph Searching

A directed **graph** consists of

- a set N of **nodes** and
- a set A of ordered pairs of nodes called **arcs**.

In this definition, a node can be anything. All this definition does is constrain arcs to be ordered pairs of nodes. There can be infinitely many nodes and arcs. We do not assume that the graph is represented explicitly; we require only a procedure that can generate nodes and arcs as needed.

The arc $\langle n_1,n_2 \rangle$ is an **outgoing arc** from n_1 and an **incoming arc** to n_2.

A node n_2 is a **neighbor** of n_1 if there is an arc from n_1 to n_2; that is, if $\langle n_1,n_2 \rangle \in A$. Note that being a neighbor does not imply symmetry; just because n_2 is a neighbor of n_1 does not mean that n_1 is necessarily a neighbor of n_2. Arcs may be **labeled**, for example, with the action that will take the agent from one state to another.

A **path** from node s to node g is a sequence of nodes $\langle n_0, n_1,..., n_k\rangle$ such that $s=n_0$, $g=n_k$, and $\langle n_{i-1},n_i\rangle \in A$; that is, there is an arc from n_{i-1} to n_i for each i. Sometimes it is useful to view a path as the sequence of arcs, $\langle n_o,n_1\rangle$, $\langle n_1,n_2\rangle$,..., $\langle n_{k-1},n_k\rangle$, or a sequence of labels of these arcs.

A **cycle** is a nonempty path such that the end node is the same as the start node - that is, a cycle is a path $\langle n_0, n_1,..., n_k\rangle$ such that $n_0=n_k$ and $k\neq 0$. A directed graph without any cycles is called a **directed acyclic graph** (**DAG**). This should probably be an **acyclic directed graph**, because it is a directed graph that happens to be acyclic, not an acyclic graph that happens to be directed, but DAG sounds better than ADG!

A **tree** is a DAG where there is one node with no incoming arcs and every other node has exactly one incoming arc. The node with no incoming arcs is called the **root** of the tree and nodes with no outgoing arcs are called **leaves**.

To encode problems as graphs, one set of nodes is referred to as the **start nodes** and another set is called the **goal nodes**. A **solution** is a path from a start node to a goal node.

Sometimes there is a **cost** - a positive number - associated with arcs. We write the cost of arc $\langle n_i,n_j\rangle$ as $cost(\langle n_i,n_j\rangle)$. The costs of arcs induces a cost of paths.

Given a path $p = \langle n_0, n_1,..., n_k\rangle$, the cost of path p is the sum of the costs of the arcs in the path:

$$cost(p) = cost(\langle n_0,n_1\rangle) + ...+ cost(\langle n_{k-1},n_k\rangle)$$

An **optimal solution** is one of the least-cost solutions; that is, it is a path p from a start node to a goal node such that there is no path p' from a start node to a goal node where $cost(p')<cost(p)$.

Example 3.4: Consider the problem of the delivery robot finding a path from location *o103* to location *r123* in the domain depicted in Figure 3.1. In this figure, the interesting locations are named. For simplicity, we consider only the locations written in bold and we initially limit the directions that the robot can travel. Figure 3.2 shows the resulting graph where the nodes represent locations and the arcs represent possible single steps between locations. In this figure, each arc is shown with the associated cost of getting from one location to the next.

In this graph, the nodes are $N=\{mail,ts,o103,b3,o109,...\}$ and the arcs are $A=\{\langle ts,mail\rangle, \langle o103,ts\rangle, \langle o103,b3\rangle, \langle o103,o109\rangle, ...\}$. Node *o125* has no neighbors. Node *ts* has one neighbor, namely *mail*. Node *o103* has three neighbors, namely *ts, b3,* and *o109*.

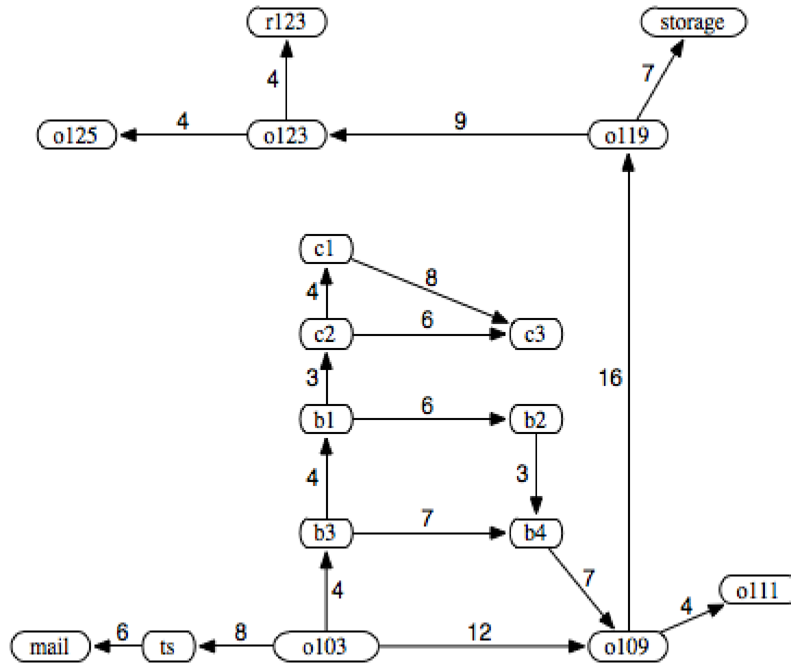


Figure : A graph with arc costs for the delivery robot domain

There are three paths from *o103* to *r123*:

⟨o103, o109, o119, o123, r123⟩

⟨o103, b3, b4, o109, o119, o123, r123⟩

⟨o103, b3, b1, b2, b4, o109, o119, o123, r123⟩

If *o103* were a start node and *r123* were a goal node, each of these three paths would be a solution to the graph-searching problem.

In many problems the search graph is not given explicitly; it is dynamically constructed as needed. All that is required for the search algorithms that follow is a way to generate the neighbors of a node and to determine if a node is a goal node.

The **forward branching factor** of a node is the number of arcs leaving the node. The **backward branching factor** of a node is the number of arcs entering the node. These factors provide measures of the complexity of graphs. When we discuss the time and space complexity of the search algorithms, we assume that the branching factors are bounded from above by a constant.

Example 3.5: In the graph of Figure 3.2, the forward branching factor of node *o103* is three; there are three arcs coming out of node *o103*. The backward branching factor of node *o103* is zero; there are no arcs coming into node *o103*. The forward branching factor of *mail* is zero and the backward branching factor of *mail* is one. The forward branching factor of node *b3* is two and the backward branching factor of *b3* is one.

The branching factor is important because it is a key component in the size of the graph. If the forward branching factor for each node is *b*, and the graph is a tree, there are b^n nodes that are *n* arcs away from any node.

A Generic Searching Algorithm

This section describes a generic algorithm to search for a solution path in a graph. The algorithm is independent of any particular search strategy and any particular graph.

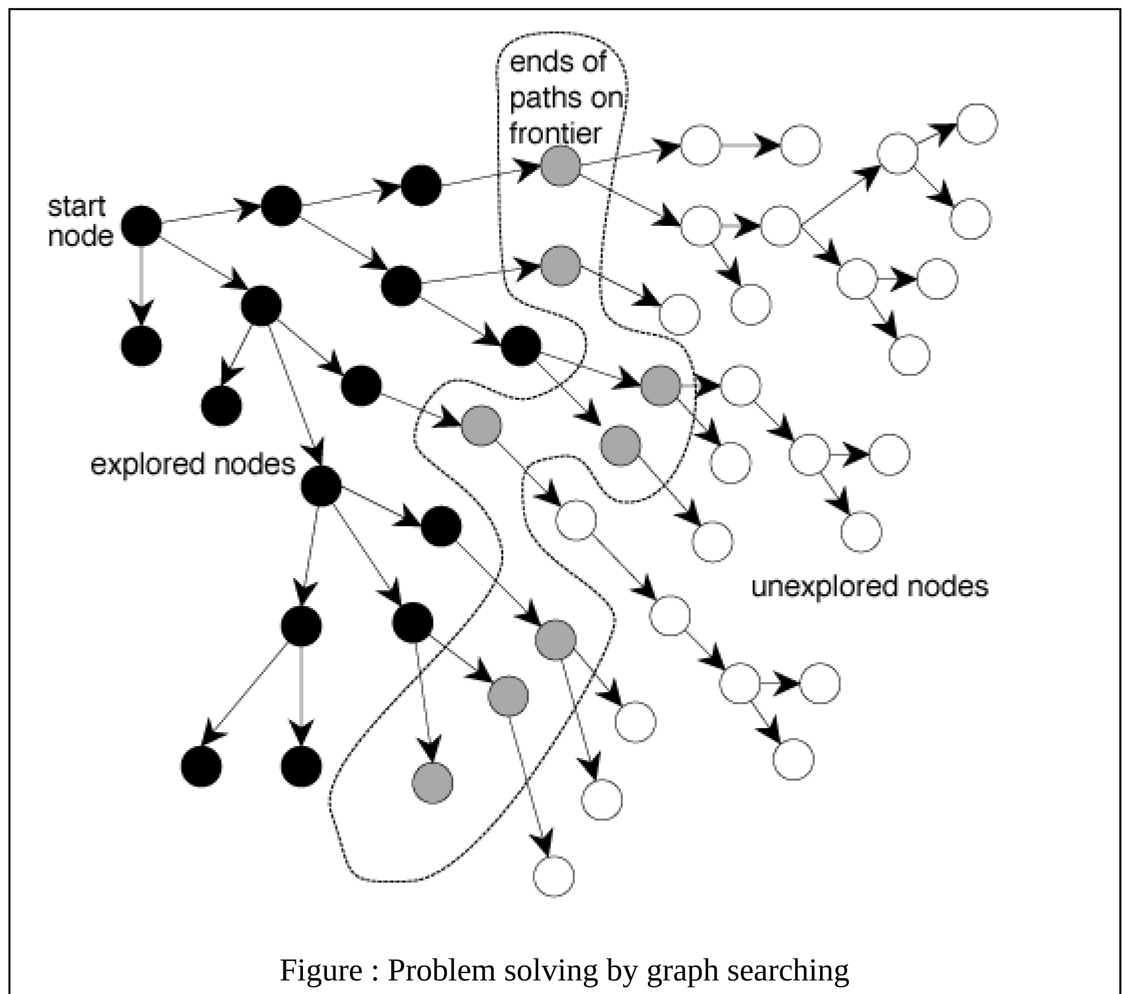


Figure : Problem solving by graph searching

The intuitive idea behind the generic search algorithm, given a graph, a set of start nodes, and a set of goal nodes, is to incrementally explore paths from the start nodes. This is done by maintaining a **frontier** (or **fringe**) of paths from the start node that have been explored. The frontier contains all of the paths that could form initial segments of paths from a start node to a goal node. (See Figure 3.3, where the frontier is the set of paths to the gray shaded nodes.) Initially, the frontier contains trivial paths containing no arcs from the start nodes. As the search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered. To expand the frontier, the searcher selects and removes a path from the frontier, extends the path with each arc leaving the last node, and adds these new paths to the frontier. A search strategy defines which element of the frontier is selected at each step.

1: **Procedure** Search(*G*,*S*,*goal*)
2: **Inputs**
3: *G*: graph with nodes *N* and arcs *A*
4: *S*: set of start nodes
5: *goal*: Boolean function of states
6: **Output**
7: path from a member of *S* to a node for which *goal* is true
8: or $\perp$ if there are no solution paths
9: **Local**
10: *Frontier*: set of paths
11: $Frontier \leftarrow \{\langle s \rangle : s \in S\}$
12: **while** ($Frontier \neq \{\}$)
13: **select** and **remove** $\langle s_0,...,s_k \rangle$ from *Frontier*
14: **if** ($goal(s_k)$) **then**
15: **return** $\langle s_0,...,s_k \rangle$
16: $Frontier \leftarrow Frontier \cup \{\langle s_0,...,s_k,s \rangle : \langle s_k,s \rangle \in A\}$
17: **return** $\perp$

Figure 3.4: Generic graph searching algorithm

The generic search algorithm is shown in Figure 3.4. Initially, the frontier is the set of empty paths from start nodes. At each step, the algorithm advances the frontier by removing a path $\langle s_0,...,s_k \rangle$ from the frontier. If $goal(s_k)$ is true (i.e., s_k is a goal node), it has found a solution and returns the path that was found, namely $\langle s_0,...,s_k \rangle$. Otherwise, the path is extended by one more arc by finding the neighbors of s_k. For every neighbor s of s_k, the path $\langle s_0,...,s_k,s \rangle$ is added to the frontier. This step is known as **expanding** the node s_k.

This algorithm has a few features that should be noted:

- The selection of a path at line 13 is non-deterministic. The choice of path that is selected can affect the efficiency; see the box for more details on our use of "select". A particular search strategy will determine which path is selected.

- It is useful to think of the *return* at line 15 as a temporary return; another path to a goal can be searched for by continuing to line 16.
- If the procedure returns ⊥, no solutions exist (or there are no remaining solutions if the proof has been retried).
- The algorithm only tests if a path ends in a goal node *after* the path has been selected from the frontier, not when it is added to the frontier. There are two main reasons for this. Sometimes a very costly arc exists from a node on the frontier to a goal node. The search should not always return the path with this arc, because a lower-cost solution may exist. This is crucial when the least-cost path is required. The second reason is that it may be expensive to determine whether a node is a goal node.

If the path chosen does not end at a goal node and the node at the end has no neighbors, extending the path means removing the path. This outcome is reasonable because this path could not be part of a path from a start node to a goal node.

Tree Search

A tree structure is a hierarchy of linked nodes where each node represents a particular state. Nodes have none, one or more child nodes. A solution is a path from the "root" node (representing the initial state) to a "goal" node (representing the desired state). Tree search algorithms attempt to find a solution by traversing the tree structure - starting at the root node and examining (expanding) the child nodes in a systematic way.

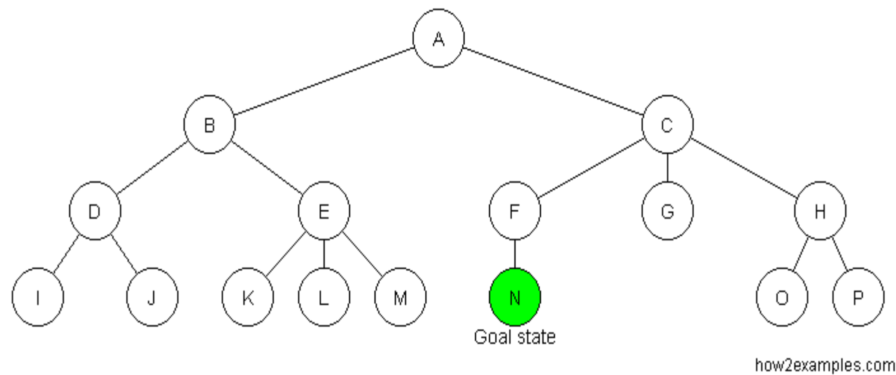
Tree search algorithms differ by the order in which nodes are traversed and can be classified into two main groups:

- *Blind search* algorithms (e.g. "Breadth-first" and "Depth-first") use a fixed strategy to methodically traverse the search tree. Blind search is not suitable for complex problems as the the large search space (number of different possible states to search) makes them impractical given time and memory constraints.
- *Best-first search* algorithms (e.g. "Greedy" and "A*") use a heuristic function to determine the order in which nodes are traversed, giving preference to states that are judged to be most likely to reach the required goal. Using a "heuristic" search strategy reduces the search space to a more manageable size.

A search strategy is *complete* if it is guaranteed to find a solution if one exists. A search strategy is *optimal* if it is guaranteed to find the best solution when several solutions exists.

Breadth-First Search

Breadth-first search starts at the root of the tree and examines all nodes at the same level before examining nodes at the next level.



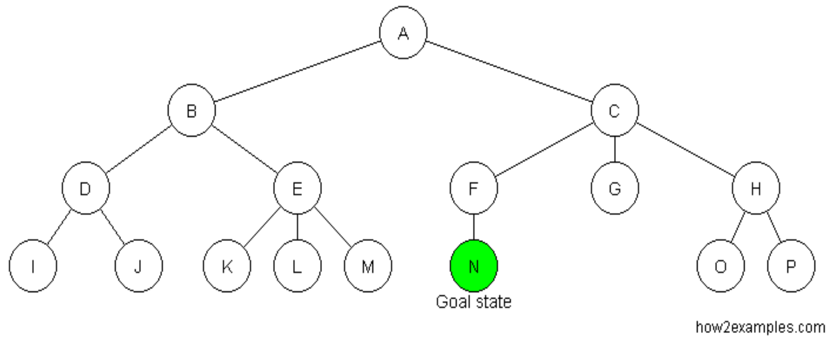
Example of Breadth First Search

As breadth-first search exhaustively examines every node at a particular depth before progressing to the next level, it is guaranteed to find the solution, if one exists, with the shortest path from the initial state. A disadvantage of breadth-first search is that it can have a high memory requirement - as a record needs to be maintained of every expanded node.

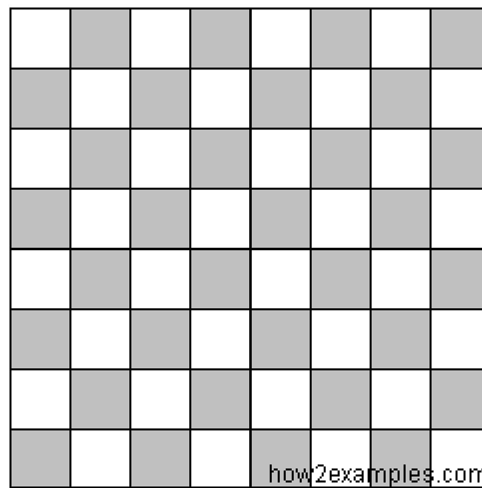
Depth-First Search

Depth-first search starts at the root node and continues down a particular path (branch) - selecting a child node at the deepest level of the tree to expand next. Only when the search hits a dead end (a node that has no child nodes) does the search "backtrack" - continuing the search from the last node it encountered whose child nodes have not been fully examined.

Unlike breadth-first search, depth-first search is not guaranteed to find the solution with the shortest path. As it is possible for depth-first search to proceed down an infinitely long branch, without ever returning to explore other branches, there is no guarantee that depth-first search will ever find a solution, even when one exists. The memory requirements of depth-first search are more modest than breadth-first search. Only a single path from the root node to the current node, plus any unexpanded nodes on the path, need to be stored.



Example of Depth First Search

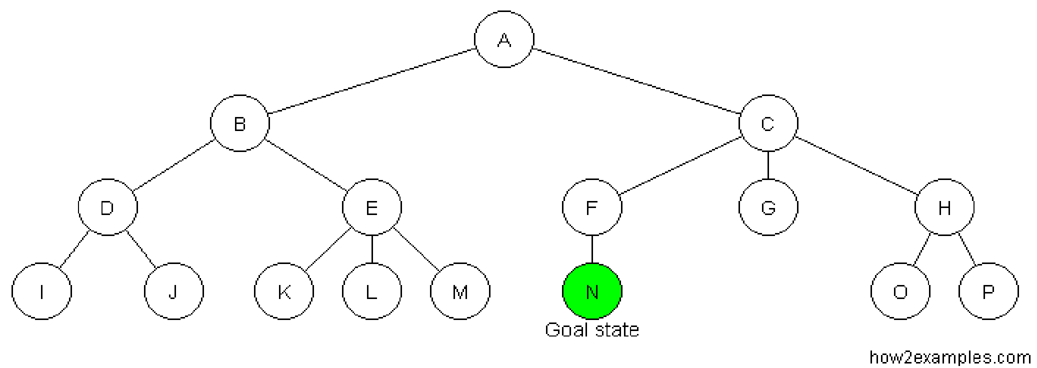


Example of solving the Eight Queens puzzle using Depth First Search

Iterative Deepening Depth-First Search

Iterative deepening depth-first search (IDDFS) operates like depth-first search - apart from that the algorithm imposes a limit on how deep the search traverses. Until a goal state is found, the search is repeated with an increased depth limit.

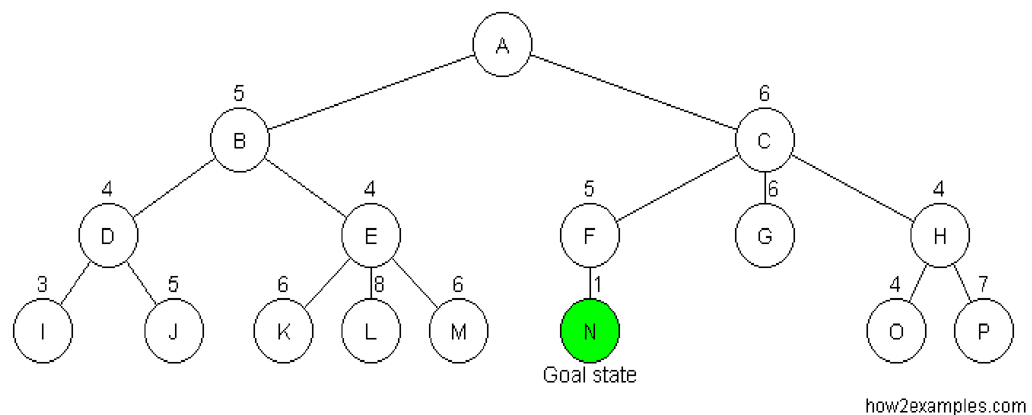
Iterative deepening depth-first search combines advantages of both breadth-first and depth-first search. By continuously incrementing the depth limit by one until a solution is found, iterative deepening depth-first search has the same strength as breadth-first search regarding always finding the shortest path to a solution. By using a depth-first approach on every iteration, iterative deepening depth-first avoids the memory cost of breadth-first search.



Example of Iterative Deepening Depth First Search

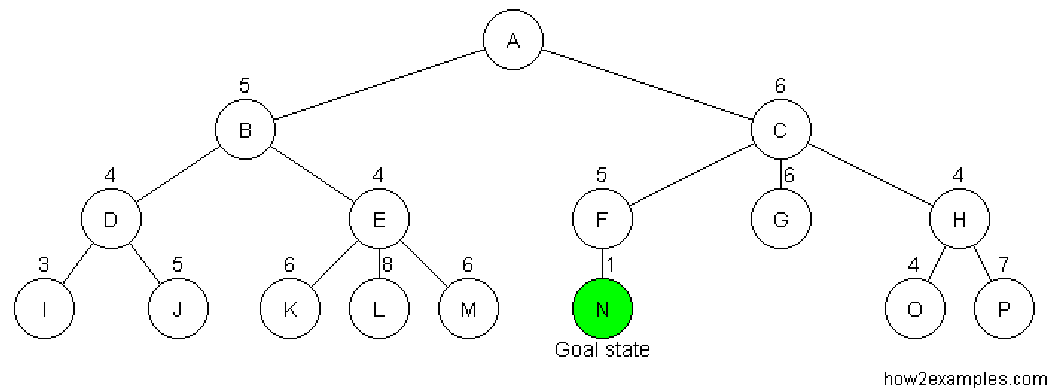
Greedy Search

Nodes are evaluated using a heuristic function. The heuristic function estimates how close a node is to the goal state. The sequence in which nodes are traversed is ordered, with the nodes considered closest to the goal state being expanded first.



Example of Greedy Search

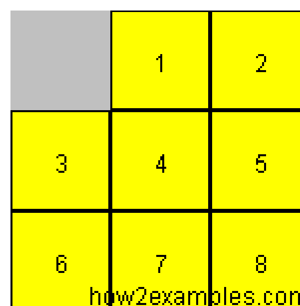
Like depth-first search, greedy search is not complete. Greedy search is not guaranteed to find the solution with the shortest path. It is possible for greedy search to proceed down an infinitely long branch without finding a solution, even when one exists.



Example of A* Search

Like breadth-first search, A* search is complete - it will always find a solution if one exists. For A* search to be optimal it must be used with an *admissibile heuristic*. An admissibile heuristic, also known as an optimistic heuristic, never overestimates the cost of reaching the goal.

When A* search reaches a goal state it has found a solution with a total cost less than or equal to the estimated cost of any unsearched paths. If the estimated costs are optimistic then the true cost of any solutions discovered by traversing the unsearched paths are guaranteed to be no better than solution already found.



Example of solving a sliding tiles puzzle using A* Search

A disadvantage of A* search is that, as it needs to maintain a list of unsearched nodes, it can require large amounts of memory. Variations of A* that require less memory include Iterative Deepening A* (IDA*) and Simplified Memory Bounded A* (SMA*).

1.8 Unit – End Exercise

1. List down the Concepts of AI
2. Explain Forward and Backward Reasoning

1.9 Answers to Check Your Progress

1. Recommendation Engine, Autonomous, NLP, Turing Test, Facial Recognition, Spam Filters, Smart Assistants, Language Translators, Voice to Text.
2. Forward Reasoning - The solution of a problem generally includes the initial data and facts in order to arrive at the solution.
   Backward Reasoning - The **backward reasoning** is inverse of forward reasoning in which goal is analysed in order to deduce the rules, initial facts and data.

1.10 Suggested Readings

1. https://en.wikipedia.org/wiki/Artificial_intelligence
2. https://www.roboticsbusinessreview.com/ai/3-basic-ai-concepts-explain-artificial-intelligence/
3. https://becominghuman.ai/introduction-to-artificial-intelligence-5fba0148ec99
4. https://analyticsindiamag.com/9-ai-concepts-every-non-technical-person-should-know/
5. https://www.dummies.com/software/other-software/5-main-approaches-ai-learning/
6. https://www.datascienceassn.org/content/four-approaches-artificial-general-intelligence
7. http://inspiratron.org/blog/2013/05/10/approaches-to-artificial-intelligence/
8. https://www.edureka.co/blog/artificial-intelligence-applications/
9. https://medium.com/kredo-ai-engineering/search-algorithms-part-1-problem-formulation-and-searching-for-solutions-28f722b7a1a6
10. https://techdifferences.com/difference-between-forward-and-backward-reasoning-in-ai.html

UNIT II PROBLEM SOLVING AGENTS

Structure

2.1 Definition

Problem Solving Agent

An agent that tries to come up with a sequence of actions that will bring the environment into a desired state.

Search

The process of looking for such a sequence, involving a systematic exploration of alternative actions.

Searching is one of the classic areas of AI.

2.2 Problems

A problem is a tuple (S,s,A,ρ,G,P)

Where S is a set of states

- $s \in S$
- is the initial state
- A is a set of actions (sometimes called operators)
- $\rho: S \times A \rightarrow S$ is a partial function, that tells you for each state, which actions will take you to which states.
- $G: S \rightarrow \text{bool}$ is the goal test function, which tells you whether a state is a goal state or not (some people would be just as happy postulating a set of goal states)
- $P: S \times (A \times S)* \rightarrow \text{real}$ is the path cost function. A path is a sequence $[s_0 a_1 s_1 a_2 s_2 \ldots a_k s_k]$ such that $\forall i \in \{1..k\} \rho(s_{i-1}, a_i) = s_i$

  - .

Example: A water jug problem

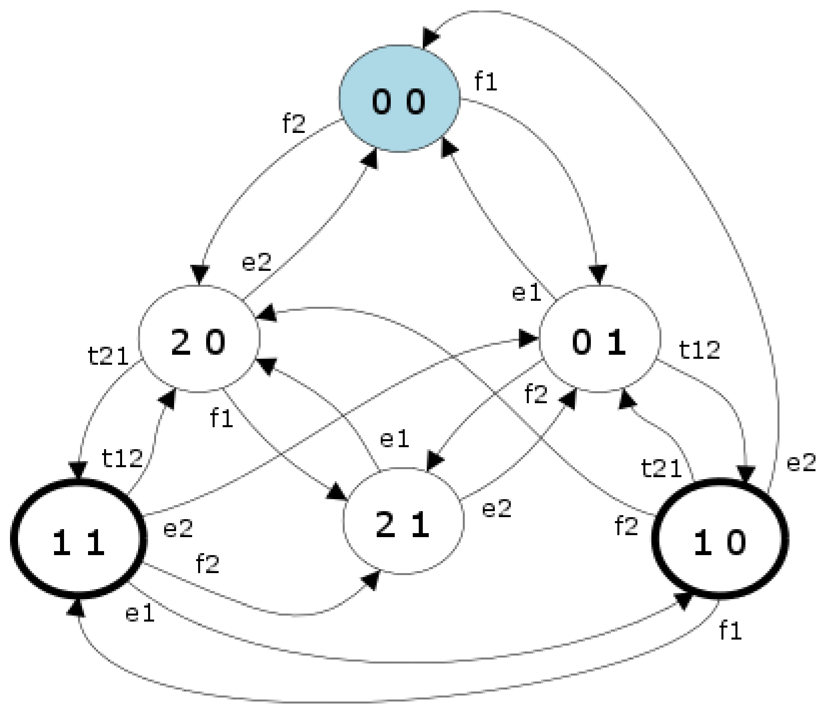
You have a two-gallon jug and a one-gallon jug; neither have any measuring marks on them at all. Initially both are empty. You need to get exactly one gallon into the two-gallon jug. Formally:

  - $S=\{(0,0),(1,0),(2,0),(0,1),(1,1),(2,1)\}$

(or, if you prefer, $\{0,1,2\} \times \{0,1\}$
- • $s=(0,0)$
- • $A=\{f2,f1,e2,e1,t21,t12\}$
- • ρ
- is given by the diagram and the table below
- $G=\lambda(x,y).x=1$
- • $P(p)=\text{length}(p)$

  - (the number of actions in the path)

A graphical view of the transition function (initial state shaded, goal states outlined bold):



And a tabular view:

	f2	e2	f1	e1	t21	t12
(0,0)	(2,0)	—	(0,1)	—	—	—
(1,0)	(2,0)	(0,0)	(1,1)	—	(0,1)	—
(2,0)	—	(0,0)	(2,1)	—	(1,1)	—
(0,1)	(2,1)	—	—	(0,0)	—	(1,0)
(1,1)	(2,1)	(0,1)	—	(1,0)	—	(2,0)
(2,1)	—	(0,1)	—	(2,0)	—	—

Example solutions

- [*f*1,*f*2,*e*2,*t*12]

  [*f*1,*e*1,*f*2,*t*21,*t*12,*f*1,*e*2,*t*12]
  [*f*2,*t*21]

There are an infinite number of solutions. Sometimes we are interested in the solution with the smallest path cost; more on this later.

Exercise: For the problem in which you have a 4-gallon jug and a 3-gallon jug, and need to get exactly two gallons in the 4-gallon jug:

- How many states are there?
- How many (legal) transitions are there?
- Solve the problem by hand

Exercise: Consider the problem of writing a method s(a, b, c) which returns the action sequence necessary to get c gallons into the jug with capacity a, given that the other jug has capacity b. Assume that a, b, and c are all **positive integers** and a > b and a > c. Does this problem always have a solution? If so, prove it; if not, give values for a, b, and c meeting the above constraints for which no solution exists.

There's this thing called the Problem Space Hypothesis, due to Newell and Simon: *All goal-oriented symbolic activity occurs in problem spaces.* If they're right, we're spending time wisely.

Exercise: What do you think of this?

Even if they're not completely right, there are still zillions of problems that can be formulated in problem spaces, e.g.

Problem	**States**	**Actions**
8-puzzle	Tile configurations	Up, Down, Left, Right
8-queens (incremental formulation)	Partial board configurations	Add queen, remove queen
8-queens (complete-state formulation)	Board configurations	Move queen
TSP	Partial tours	Add next city, pop last city
Theorem Proving	Collection of known theorems	Rules of inference

Vacuum World	Current Location and status of all rooms	Left, Right, Suck
Road Navigation (Route Finding)	Intersections	Road segments
Internet Searching	Pages	Follow link
Counterfeit Coin Problem	A given weighing	Outcome of the weighing (less, equal, greater)

2.3 Problem Types

State Finding vs. Action Sequence Finding

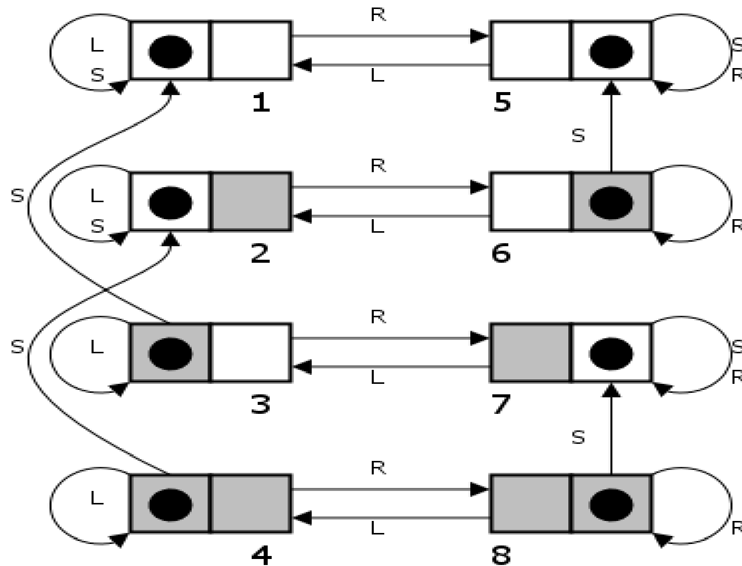
A fundamental distinction:

Action Sequence Finding	State Finding
We know the state space in advance. We know which states are goals. We have to find the sequence of actions that get us to a goal state. The sequence may be contingent, or expressed as an AND-OR tree, but the actions matter.	We only know the properties that a goal state should have, but we don't even know if any goal states exist. We just need to find a state that satisfies certain constraints! We don't care what action sequence gets us there.
Optimality is concerned with "cheapest path"	Optimality is concerned with the "best state"
Examples: 8-puzzle, water jug, vacuum world, route navigation, games, many robotics problems	Examples: N-queens, integrated circuit layout, factory floor layout, job-shop scheduling, automatic programming, portfolio management, network optimization, most other kinds of optimization problems

Offline vs. Online Problems

In an online problem, the agent doesn't even know what the state space is, and has to build a model of it as it acts. In an offline problem, percepts don't matter at all. **An agent can figure out the entire action sequence before doing anything at all**.

Offline Example: Vacuum World with two rooms, cleaning always works, a square once cleaned stays clean. States are 1 – 8, goal states are 1 and 5.



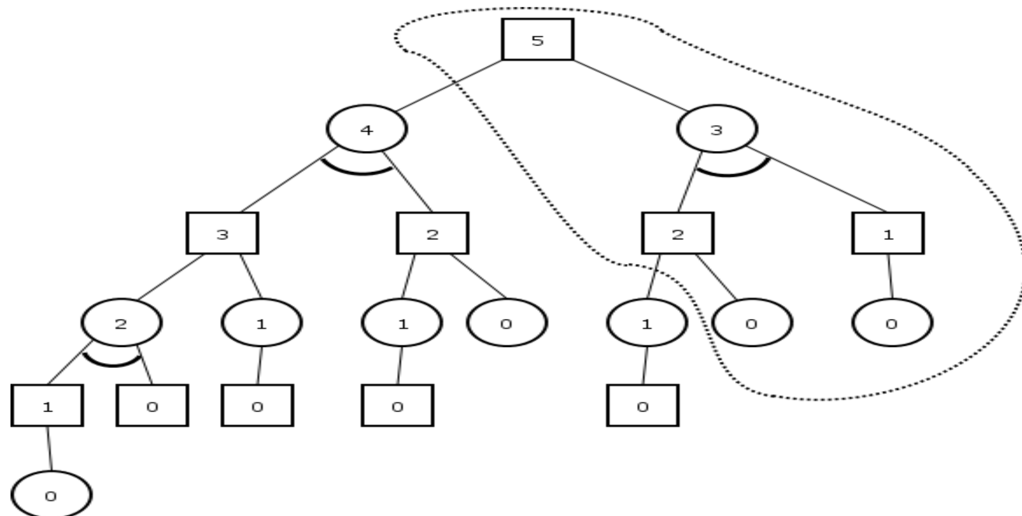
Exercise: Determine the cheapest solution, starting in state 4, for a vacuum agent assuming each suck costs 2, and each movement costs 1.

Sensorless (Conformant) Problems

The agent doesn't know where it is. We can use *belief states* (sets of states that the agent might be in). Example from above deterministic, static, single-agent vacuum world:

In State	Left	Right	Suck
12345678	1234	5678	1257
1234	1234	5678	12
5678	1234	5678	57
1257	123	567	1257
12	12	56	12
57	13	57	57
123	123	567	12
567	123	567	57
56	12	56	5
13	13	57	1
5	1	5	5
1	1	5	1

Note the goal states are 1 and 5. If a state 15 was reachable, it would be a goal too.



In general then, a solution is a subtree in which

- The root node is in the subtree
- For every OR node in the subree, at least one child is in the subtree
- For every AND node in the subtree, all children are in the subtree
- All leaves are goal states

If the tree has only OR nodes, then the solution is just a path.

2.4 Search Algorithms

---

1. **Strategies**
2. Hey, we know what a problem is, what a problem space is, and even what a solution is, but *how exactly do we search the space*? Well there are zillions of approaches:
3. breadth-first, uniform-cost
4. depth-first, backtracking, depth-limited, depth-first iterative deepening
5. backwards chaining, bidirectional search

- greedy best-first, A*, IDA*, RBFS, IE, MA*, SMA*
- hill-climbing (stochastic, first-choice, random-restart), random walk
- simulated annealing, beam search, genetic algorithms
- LRTA*

2.4.1 Types of Problem-Solving Tasks

Agents may be asked to be

- Satisficing — find any solution
- Optimizing — find the best (cheapest) solution
- Semi-optimizing — find a solution close to the optimal

An algorithm is

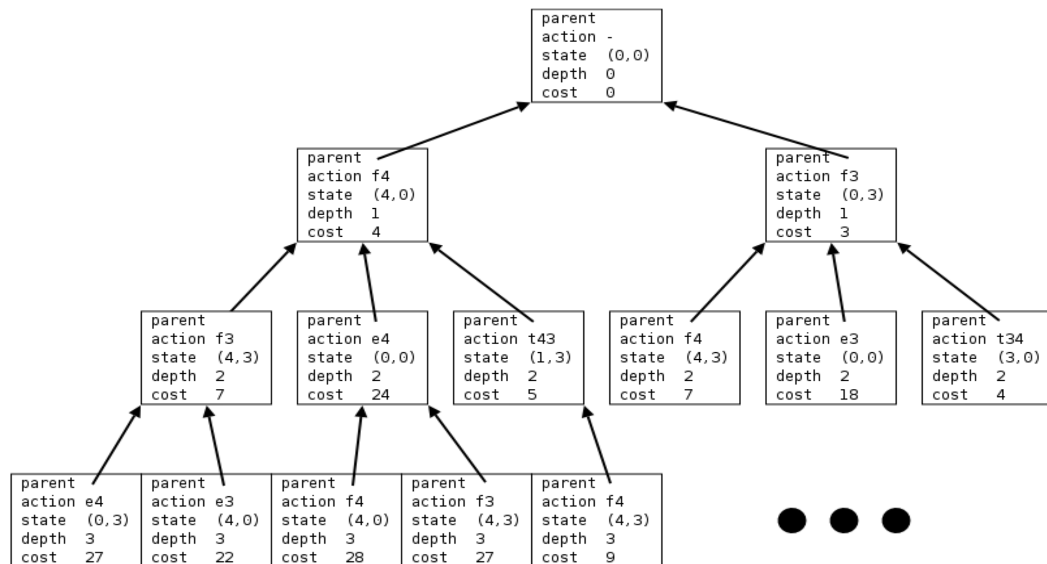
- *Complete*, if will find a solution if one exists
- *Optimal*, if it finds the cheapest solution

Search Trees

Search algorithms generate a search tree on the fly. Search trees contain *nodes*. Each node in the tree contains information about a particular path in the problem space. **Nodes are not the same thing as states.**

```
class Node {
   State state;
   Action action;
   Node parent;
   int depth;
   int cost;
   Node(State state, Action action, Node parent, int stepCost) {
      this.state = state;
      this.action = action;
      this.parent = parent;
      depth = parent == null ? 0 : parent.depth + 1;
      cost = parent == null ? 0 : parent.cost + stepCost;
   }
}
```

Example: The water jug problem with 4 and 3 gallon jugs. Cost is 1 point per gallon used when filling, 1 point to make a transfer, 5 points per gallon emptied (since it makes a mess). The search tree might start off like this:



Generate

Compute a new search tree node from its parent

Expand

Generate, from a node, all of its children

Search trees have

Branching Factor (b)

The average number of children of a node

Depth (d)

Height of the shortest solution subtree

Max Depth (m)

Maximum depth of the search tree (can be infinite)

The complexity of most search algorithms can be written as a function of one or more of b, d and m

Complexity

- The time complexity has to do with to the number of nodes generated.
- The space complexity has to do with the number of nodes that have to be stored (at a time)

1. In general though there may be more states than there are fundamental particles in the universe. But we need to find a solution. Usually is helpful to
2. Find ways to identify large subsets of states that could never possibly be goal states so you don't have to ever visit them.
3. Don't *re*visit states

2.5 Measuring Problem-Solving Performances

Search as a black box will result in an output that is either **failure** or a **solution**, We will evaluate a search algorithm`s performance in four ways:

1. Completeness: is it guaranteed that our algorithm always finds a solution when there is one ?
2. Optimality: Does our algorithm always find the optimal solution ?
3. Time complexity: How much time our search algorithm takes to find a solution ?
4. Space complexity: How much memory required to run the search algorithm?

Time and Space in complexity analysis are measured with respect to the number of nodes the problem graph has in terms of asymptotic notations.

In AI, complexity is expressed by three factors ***b***, ***d*** and **m:**

1. *b* the **branching factor** is the maximum number of successors of any node.
2. *d* the **depth** of the deepest goal.
3. *m* the maximum **length** of any path in the state space.

Check your Progress
Note: a. Write your answer in the space given below
b. Compare your answer with those given at the end of the unit.

i. List the types of Problem-Solving Tasks

…………………………………………………………………………………

…………………………………………………………………………………

2.6 Unit – End Exercise

1. Define Problem-Solving Tasks
2. How to measure Problem-Solving Performance
3. List the types of problem-solving tasks.

2.7 Answers to Check Your Progress

1. Satisfycing, optimizing, semi-optimizing. The algorithms are complete and optimal
2. Completeness, Optimality, time Complexity and Space Complexity
3. Agents may be asked to be: Satisficing — find any solution, Optimizing — find the best (cheapest) solution, Semi-optimizing — find a solution close to the optimal

2.8 Suggested Readings

1. https://www.edureka.co/blog/artificial-intelligence-applications/
2. https://medium.com/kredo-ai-engineering/search-algorithms-part-1-problem-formulation-and-searching-for-solutions-28f722b7a1a6
3. https://www.geeksforgeeks.org/search-algorithms-in-ai/
4. https://artint.info/html/ArtInt_83.html
5. https://guttulus.com/what-is-local-search-algorithm-in-artificial-intelligence/

UNIT III SEARCH STRATEGIES

Structure

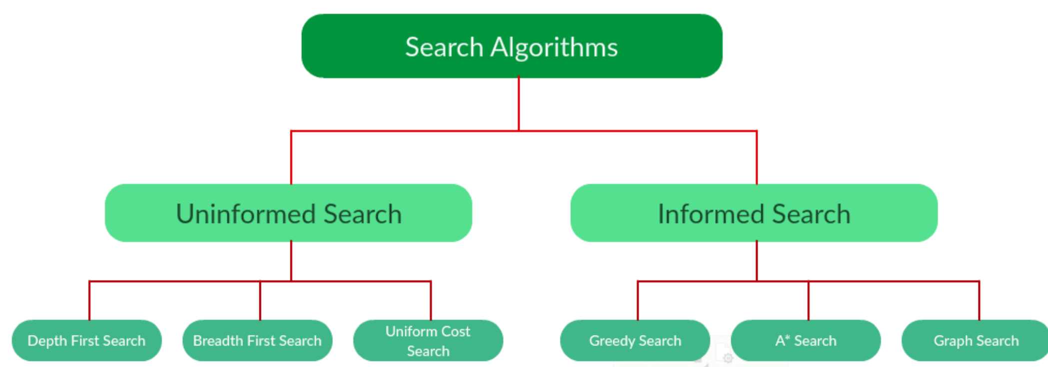
Search Algorithms in AI

Artificial Intelligence is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.

- A search problem consists of:
    - **A State Space.** Set of all possible states where you can be.
    - **A Start State.** The state from where the search begins.
    - **A Goal Test.** A function that looks at the current state returns whether or not it is the goal state.
- The **Solution** to a search problem is a sequence of actions, called the **plan** that transforms the start state to the goal state.
- This plan is achieved through search algorithms.

Types of search algorithms

There are far too many powerful search algorithms out there to fit in a single article. Instead, this article will discuss *six* of the fundamental search algorithms, divided into *two* categories, as shown below.



Note that there is much more to search algorithms that the chart I have provided above. However, this article will mostly stick to the above chart, exploring the algorithms given there.

Uninformed Search Algorithms

The search algorithms in this section have no additional information on the goal node other than the one provided in the problem definition. The plans to reach the goal state from the start state differ only by the order and/or length of actions. Uninformed search is also called **Blind search**.

The following uninformed search algorithms are discussed in this section.

1. Depth First Search
2. Breath First Search
3. Uniform Cost Search

Each of these algorithms will have:

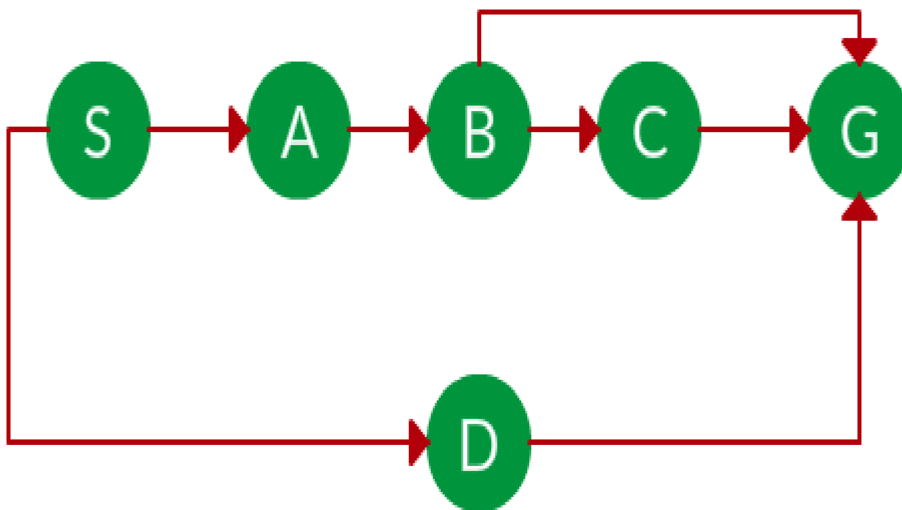
- A problem **graph,** containing the start node S and the goal node G.
- A **strategy,** describing the manner in which the graph will be traversed to get to G .
- A **fringe,** which is a data structure used to store all the possible states (nodes) that you can go from the current states.
- A **tree,** that results while traversing to the goal node.
- A solution **plan,** which the sequence of nodes from S to G.

Depth First Search

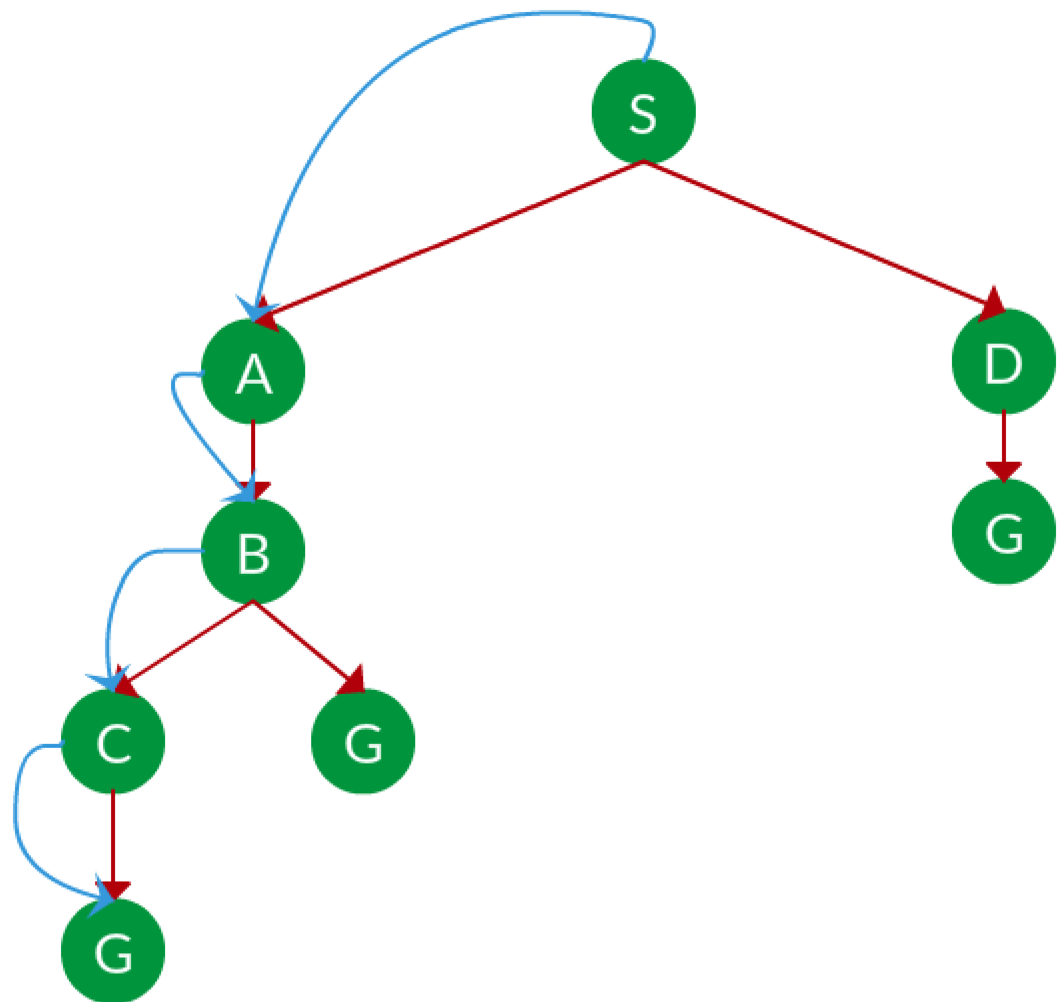
Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Example:

Question. Which solution would DFS find to move from node S to node G if run on the graph below?



Solution. The equivalent search tree for the above graph is as follows. As DFS traverses the tree "deepest node first", it would always pick the deeper branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



Path: S-> A->B->C->G

Let S=the depth of the search tree=number of levels of the search tree

= number of nodes in level

Time complexity: equivalent to the number of nodes traversed in DFS

Space Complexity: equivalent to how large can the fringe get.

Completeness: DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.

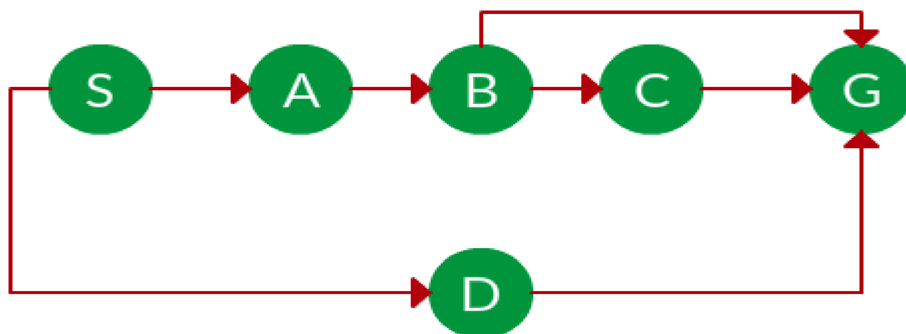
Optimality DFS is not optimal, meaning the number of steps in reaching the solutions, or the cost spent in reaching it is high.

<u>Breadth First Search</u>

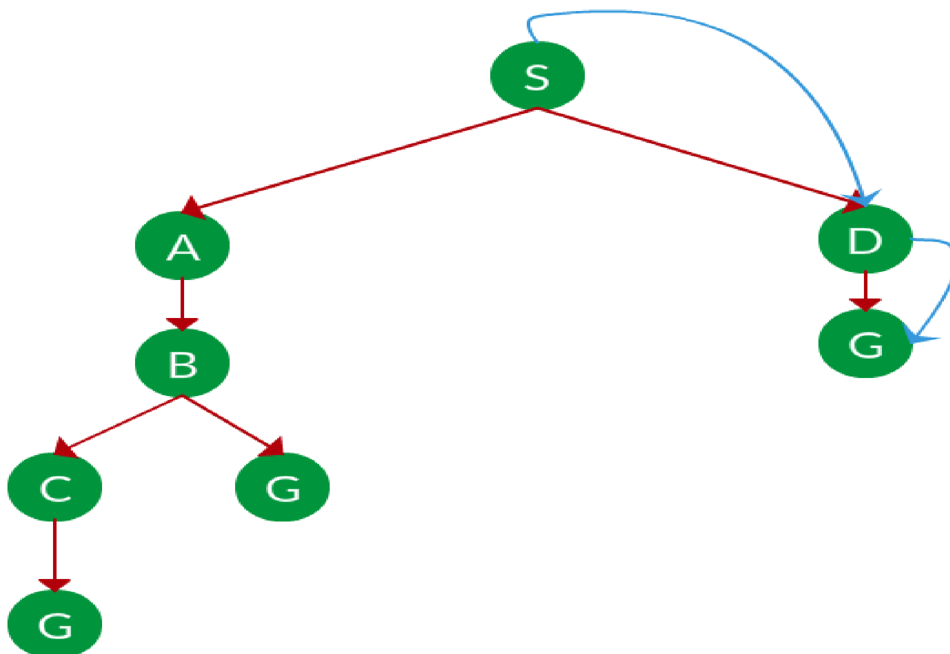
Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

Example:

Question. Which solution would BFS find to move from node S to node G if run on the graph below?



Solution. The equivalent search tree for the above graph is as follows. As BFS traverses the tree "shallowest node first", it would always pick the shallower branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



Path: S->D->G

Let =the depth of the shallowest solution

= number of nodes in the level

Time complexity: equivalent to the number of nodes traversed in BFS until the shallowest solution.

Space Complexity: Equivalent to how large can the fringe get.

Completeness: BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists

Optimality: BFS is optimal as long as the cost of all edges aare equal

Uniform Cost Search

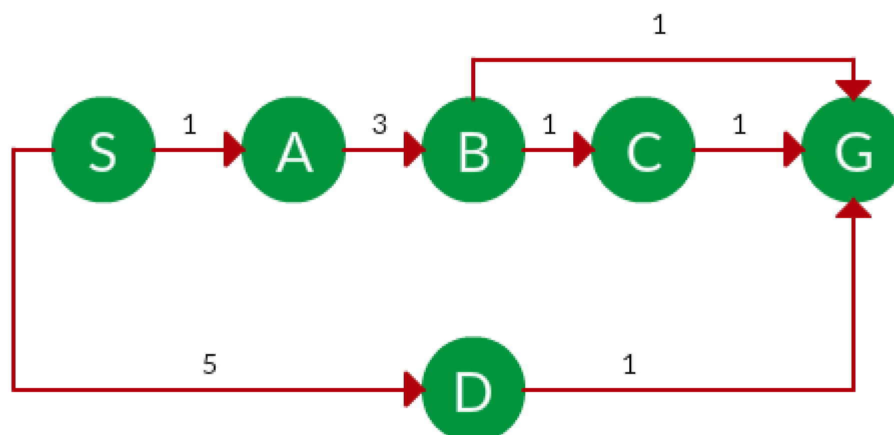
UCS is different from BFS and DFS because here the costs come into play. In other words, traversing via different edges might not have the same cost. The goal is to find a path where the cumulative sum of costs is least.

Cost of a node is defined as:

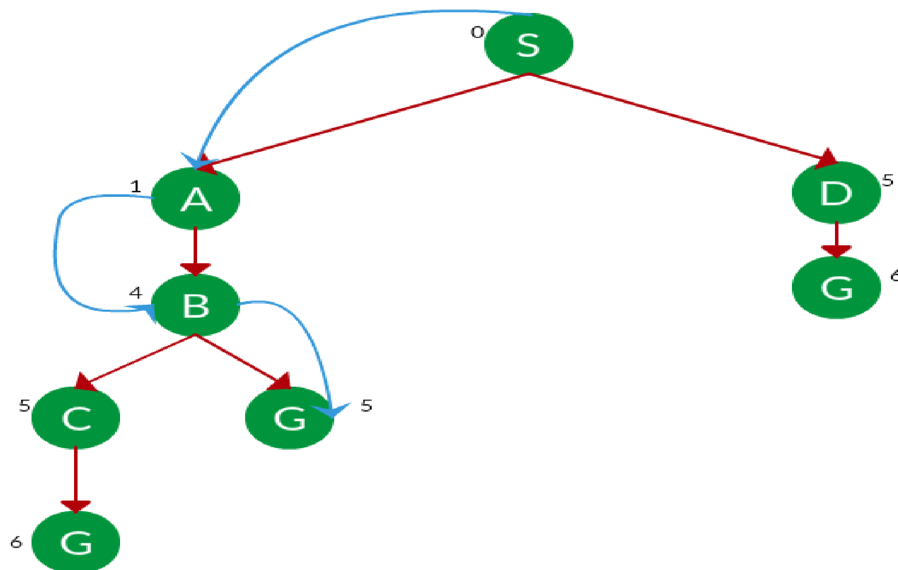
cost(node) = cumulative cost of all nodes from root
cost(root) = 0

Example:

Question. Which solution would UCS find to move from node S to node G if run on the graph below?



Solution. The equivalent search tree for the above graph is as follows. Cost of each node is the cumulative cost of reaching that node from the root. Based on UCS strategy, the path with least cumulative cost is chosen. Note that due to the many options in the fringe, the algorithm explores most of them so long as their cost is low, and discards them when a lower cost path is found; these discarded traversals are not shown below. The actual traversal is shown in blue.



Path: S->A->B->G

Cost: 5

Let = cost of solution

= area cost

Then effective depth

Advantages:

- UCS is complete.
- UCS is optimal.

Disadvantages:

- Explores options in every “direction”.
- No information on goal location.

3.4 Informed Search

Informed Search Algorithms

Here, the algorithms have information on the goal state, which helps in more efficient searching. This information is obtained by something called a *heuristic.*

In this section, we will discuss the following search algorithms.

1. Greedy Search
2. A* Tree Search
3. A* Graph Search

Search Heuristics: In an informed search, a heuristic is a *function* that estimates how close a state is to the goal state. For examples – Manhattan distance, Euclidean distance, etc. (Lesser the distance, closer the goal.) Different heuristics are used in different informed algorithms discussed below.

Greedy Search

In greedy search, we expand the node closest to the goal node. The “closeness” is estimated by a heuristic h(x) .

Heuristic: A heuristic h is defined as-

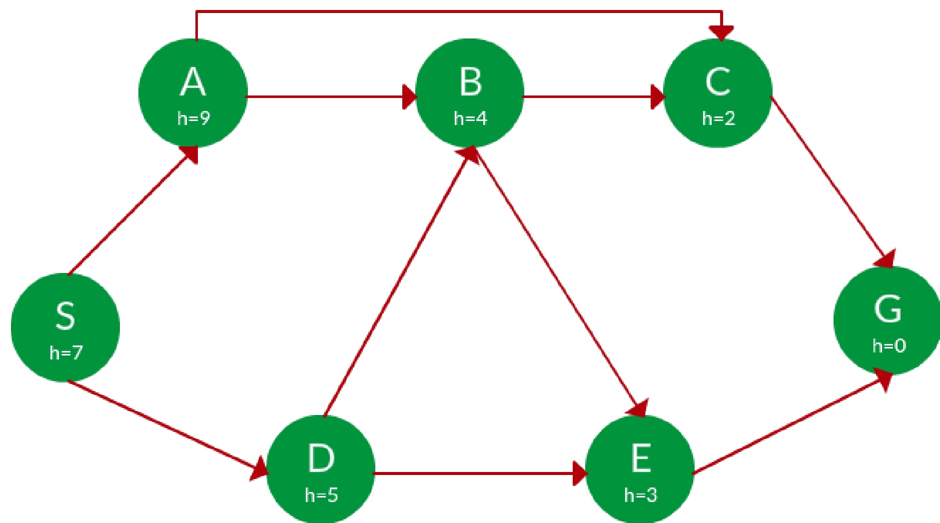
h(x) = Estimate of distance of node x from the goal node.

Lower the value of h(x), closer is the node from the goal.

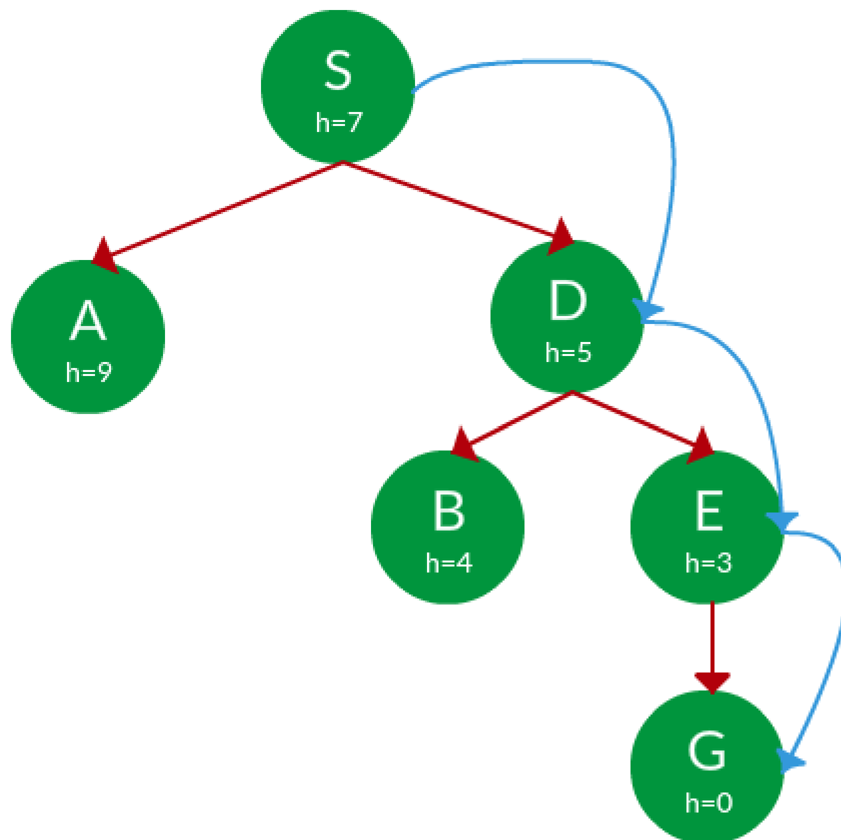
Strategy: Expand the node closest to the goal state, *i.e.* expand the node with lower h value.

Example:

Question. Find the path from S to G using greedy search. The heuristic values h of each node below the name of the node.



Solution. Starting from S, we can traverse to A(h=9) or D(h=5). We choose D, as it has the lower heuristic cost. Now from D, we can move to B(h=4) or E(h=3). We choose E with lower heuristic cost. Finally, from E, we go to G(h=0). This entire traversal is shown in the search tree below, in blue.



Path: S->D->E->G

Advantage: Works well with informed search problems, with fewer steps to reach a goal.
Disadvantage: Can turn into unguided DFS in the worst case.

A* Tree Search

A* Tree Search, or simply known as A* Search, combines the strengths of uniform-cost search and greedy search. In this search, the heuristic is the summation of the cost in UCS, denoted by g(x), and the cost in greedy search, denoted by h(x). The summed cost is denoted by f(x).

Heuristic: The following points should be noted wrt heuristics in A* search.

- Here, h(x) is called the **forward cost**, and is an estimate of the distance of the current node from the goal node.

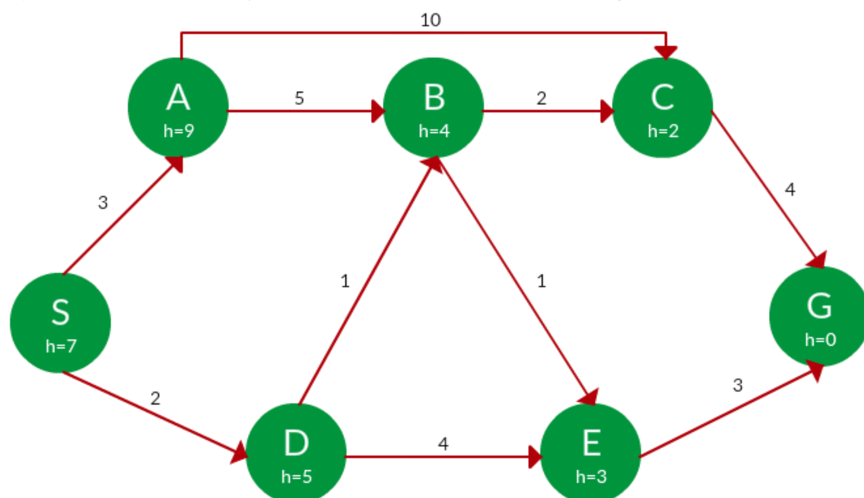
- And, g(x) is called the **backward cost**, and is the cumulative cost of a node from the root node.

- A* search is optimal only when for all nodes, the forward cost for a node h(x) underestimates the actual cost h*(x) to reach the goal. This property of A* heuristic is called **admissibility**.

Strategy: Choose the node with lowest f(x) value.

Example:

Question. Find the path to reach from S to G using A* search.



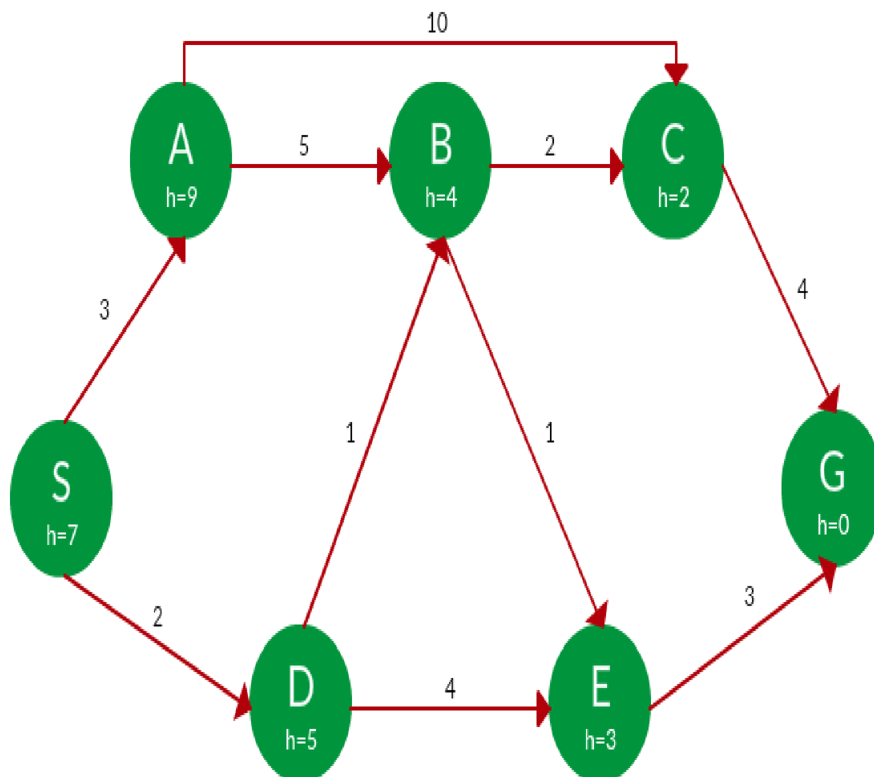
1. **Solution.** Starting from S, the algorithm computes g(x) + h(x) for all nodes in the fringe at each step, choosing the node with the lowest sum. The entire working is shown in the table below.
2. Note that in the fourth set of iteration, we get two paths with equal summed cost f(x), so we expand them both in the next set. The path with lower cost on further expansion is the chosen path.

Path	**h(x)**	**g(x)**	**f(x)**
S	7	0	7
S -> A	9	3	12
S -> D	5	2	7
S -> D -> B	4	2 + 1 = 3	7
S -> D -> E	3	2 + 4 = 6	9
S -> D -> B -> C	2	3 + 2 = 5	7
S -> D -> B -> E	3	3 + 1 = 4	7
S -> D -> B -> C -> G	0	5 + 4 = 9	9
S -> D -> B -> E -> G	0	4 + 3 = 7	7

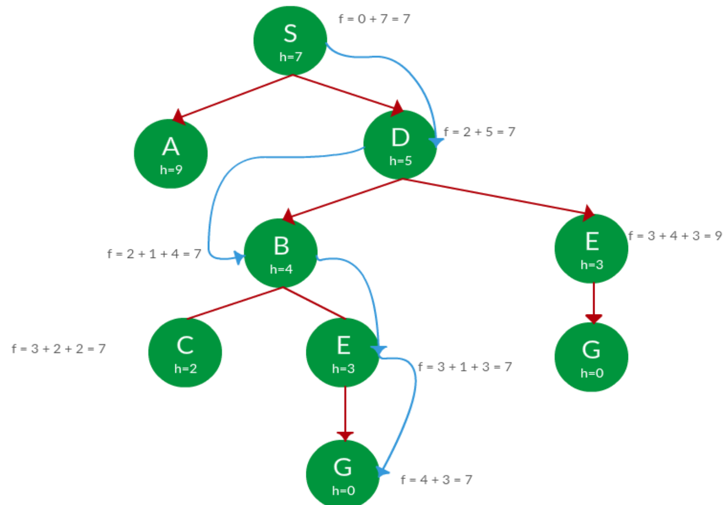
A* Graph Search

- A* tree search works well, except that it takes time re-exploring the branches it has already explored. In other words, if the same node has expanded twice in different branches of the search tree, A* search might explore both of those branches, thus wasting time
- A* Graph Search, or simply Graph Search, removes this limitation by adding this rule: **do not expand the same node more than once.**
- **Heuristic.** Graph search is optimal only when the forward cost between two successive nodes A and B, given by h(A) - h (B) , is less than or equal to the backward cost between those two nodes g(A -> B). This property of graph search heuristic is called **consistency**.

Question. Use graph search to find path from S to G in the following graph.



Solution. We solve this question pretty much the same way we solved last question, but in this case, we keep a track of nodes explored so that we don't re-explore them.



Explored Nodes

S

D

B

E

G

- Path: S -> D ->B -> C ->E -> G
- Cost: 7

Check your Progress Note: a. Write your answer in the space given below b. Compare your answer with those given at the end of the unit. i. Define Uniform Cost Search …………………………………………………………………………………… …… …………………………………………………………………………………… ……

3.5 Local Search Algorithms

The previous sections have considered algorithms that systematically search the space. If the space is finite, they will either find a solution or report that no solution exists. Unfortunately, many search spaces are too big for systematic search and are possibly even infinite. In any reasonable time, systematic search will have failed to consider enough of the search space to give any meaningful results. This section and the next consider methods intended to work in these very large spaces. The methods do not systematically search the whole search space but they are designed to find solutions quickly on average. They do not guarantee that a solution will be found even if one exists, and so they are not able to prove that no solution exists. They are often the method of choice for applications where solutions are known to exist or are very likely to exist.

Local search methods start with a complete assignment of a value to each variable and try to iteratively improve this assignment by improving steps, by taking random steps, or by restarting with another complete assignment. A wide variety of local search techniques has been proposed. Understanding when these techniques work for different problems forms the focus of a number of research communities, including those from both operations research and AI.

```
Procedure Local-Search(V,dom,C)
    Inputs
        V: a set of variables
        dom: a function such that dom(X) is the domain of variable X
        C: set of constraints to be satisfied
    Output
        complete assignment that satisfies the constraints
    Local
        A[V] an array of values indexed by V
    repeat
        for each variable X do
            A[X] ←a random value in dom(X);
        while (stopping criterion not met & A is not a satisfying assignment)
            Select a variable Y and a value V ∈dom(Y)
            Set A[Y] ←V
        if (A is a satisfying assignment) then
            return A
    until termination
```

The generic local search algorithm for CSPs is given in Figure 4.6. *A* specifies an assignment of a value to each variable. The first *for each* loop assigns a random value to each variable. The first time it is executed is called a **random initialization**. Each iteration of the outer loop is called a **try**. A common way to implement a new try is to do a **random restart**. An alternative to random initialization is to use a construction heuristic that guesses a solution, which is then iteratively improved.

The while loop does a **local search**, or a **walk**, through the assignment space. It maintains a current assignment *S*, considers a set of **neighbors** of the current assignment, and selects one to be the next current assignment. In Figure 4.6, the neighbors of a total assignment are those assignments that differ in the assignment of a single variable. Alternate sets of neighbors can also be used and will result in different search algorithms.

This walk through assignments continues until either a satisfying assignment is found and returned or some stopping criterion is satisfied. The stopping criterion is used to decide when to stop the current local search and do a random restart, starting again with a new assignment. A stopping criterion could be as simple as stopping after a certain number of steps.

This algorithm is not guaranteed to halt. In particular, it goes on forever if there is no solution, and it is possible to get trapped in some region of the search space. An algorithm is **complete** if it finds an answer whenever there is one. This algorithm is incomplete.

One instance of this algorithm is **random sampling**. In this algorithm, the stopping criterion is always true so that the while loop is never executed. Random sampling keeps picking random assignments until it finds one that satisfies the constraints, and otherwise it does not halt. Random sampling is complete in the sense that, given enough time, it guarantees that a solution will be found if one exists, but there is no upper bound on the time it may take. It is very slow. The efficiency depends only on the product of the domain sizes and how many solutions exist.

Another instance is a **random walk**. In this algorithm the while loop is only exited when it has found a satisfying assignment (i.e., the stopping criterion is always false and there are no random restarts). In the while loop it selects a variable and a value at random. Random walk is also complete in the same sense as random sampling. Each step takes less time than resampling all variables, but it can take more steps than random sampling, depending on how the solutions are distributed. Variants of this algorithm are applicable when the domain sizes of the variables differ; a random walk algorithm can either select a variable at random and then a value at random, or select a variable-value pair at random. The latter is more likely to select a variable when it has a larger domain.

Artificial Intelligence is up there with the discovery of electricity in terms of revolutionary discoveries. Although its discovery wasn't as spontaneous as that of electricity, AI's impact promises to be as widespread as that of electricity. A lot of the operations, including major aspects of search engines. In this blog we want to take a quick look how AI works and the significance of algorithms in AI.

Artificial intelligence is the art of imbuing machines with human like intelligence to enable them to carry tasks which are otherwise a reserve for humans. This is done through repeated learning where algorithms are used to discover patterns and generate solutions based on the data that the algorithms have been exposed to.

This generation of solutions and ability to predict future occurrences is dependent on the algorithms ability to come up with the right solutions when called upon. This is why it is important to understand the workings of the various types of algorithms used in AI and local search algorithm is one of the most important and frequently used algorithm in AI. Before looking at what local search algorithm is, what is a search algorithm?

Definition of Search Algorithms

An algorithm is a set of formulas and commands which are used to solve problems and yield possible solutions to problems which people are presented with. Searching on the other hand is the universal method of solving problems in AI which is the premise on which search engines are built.

Search algorithms are therefore the set of conditions and rules which are used to find a solution within a problem space. The problem space being the environment which the searching takes place.

There are different types of Search algorithms but in our context, we are concerned with the local search algorithm. In SEO a lot of emphasis is paid to the local search algorithm because it entirely is the guiding light when it comes to providing results in the search engine results pages.

Local Search Algorithm

To provide the best possible result, local search algorithms move iteratively from one possible solution to a neighbor solution and so on until the best possible set of results is achieved. The basis of the local search algorithm is that to any presented problem, there are multiple results and the tricky part is obtaining the best possible or the locally optimal result.

To obtain the locally optimal result, the algorithm keeps picking up solutions and their neighbors until there are no more improving configurations in the neighborhood. At the point where there are no improving configurations in the neighborhood, then the local search is stuck at a point known as a locally optimal point.

To curb the tendency of searches hitting the locally optimal points you can make use of any of the following methods: Restarts where you conduct the search over and over again but slightly tweaking the initial conditions or alternatively using iterated local search which is more complex.

How the searching comes to a stop

So, how does the local search arrive at its best solution? At what point is the search terminated? Termination of local search is done in any of the two ways below;

Definite time of search

In the time bound termination, the algorithm presents the best possible results within a stipulated amount of time. Any other result beyond the stipulated time of running of the algorithm is not deemed to be appropriate enough unless, there is need to improve the scope of the results.

When the right answer is arrived at

The other method through which termination is arrived at in Local search is when the best solution is arrived at. If the solution can’t further be improved in any given number of steps or searches then the search is ended.

Properties of Local Search Algorithm in Artificial Intelligence

Indefinite time

In AI, the local search algorithm is also referred to as the anytime algorithm because it always will output a solution even if it is interrupted before the defined period of searching elapses. The results which will be outputted at this point will all be valid solutions.

Approximation algorithms

The other attribute of approximation algorithm is the fact that it is indefinite in nature it doesn’t output definite results but approximations. This basically means that the solutions that are outputted at any given point are not the best possible solutions but rather the solutions that meet the search criteria or are in the neighborhood of the correct solution.

3.6 Optimization Techniques

Instead of just having possible worlds satisfy constraints or not, we often have a **preference** relation over possible worlds, and we want a best possible world according to the preference. The preference is often to minimize some error.

An **optimization problem** is given

- a set of variables, each with an associated domain;
- an **objective function** that maps total assignments to real numbers; and
- an **optimality criterion**, which is typically to find a total assignment that minimizes or maximizes the objective function.

The aim is to find a total assignment that is optimal according to the optimality criterion. For concreteness, we assume that the optimality criterion is to minimize the objective function.

A **constrained optimization problem** is an optimization problem that also has hard constraints specifying which variable assignments are possible. The aim is to find a best assignment that satisfies the hard constraints.

A huge literature exists on optimization. There are many techniques for particular forms of constrained optimization problems. For example, linear programming is the class of constrained optimization where the variables are real valued, the objective function is a linear function of the variables, and the hard constraints are linear inequalities. We do not cover these specific techniques. Although they have many applications, they have limited applicability in the space of all optimization problems. We do cover some general techniques that allow more general objective functions. However, if the problem you are interested in solving falls into one of the classes for which there are more specific algorithms, or can be transformed into one, it is generally better to use those techniques than the general algorithms presented here.

In a **constraint optimization problem**, the objective function is factored into a set of functions of subsets of the variables called soft constraints. A **soft constraint** assigns a cost for each assignment of values to some subset of the variables. The value of the objective function on a total assignment is the sum of the costs given by the soft constraints on that total assignment. A typical optimality criterion is to minimize the objective function.

Like a hard constraint, a soft constraint has a **scope** that is a set of variables. A soft constraint is a function from the domains of the variables in its scope into a real number, called its **evaluation**. Thus, given an assignment of a value to each variable in its scope, this function returns a real number.

Example 4.29: Suppose a number of delivery activities must be scheduled, similar to Example 4.8, but, instead of hard constraints, there are preferences on times for the activities. The soft constraints are costs associated with combinations of times. The aim is to find a schedule with the minimum total sum of the costs.Suppose variables *A*, *C*, *D*, and *E* have domain *{1,2}*, and variable *B* has domain *{1,2,3}*. The soft constraints are c_1:

A	*B*	Cost
1	1	5
1	2	2
1	3	2
2	1	0
2	2	4
2	3	3

B	*C*	Cost
1	1	5
1	2	2
2	1	0
2	2	4
3	1	2
3	2	0

c_3:

B	D	Cost
1	1	3
1	2	0
2	1	2
2	2	2
3	1	2
3	2	4

Thus, the scope of c_1 is $\{A,B\}$, the scope of c_2 is $\{B,C\}$, and the scope of c_3 is $\{B,D\}$. Suppose there are also constraints $c_4(C,E)$ and $c_5(D,E)$.

An assignment to some of the variables in a soft constraint results in a soft constraint that is a function of the other variables. In the preceding example, $c_1(A=1)$ is a function of B, which when applied to $B=2$ evaluates to 2.

Given a total assignment, the evaluation of the total assignment is the sum of the evaluations of the soft constraints applied to the total assignment. One way to formalize this is to define operations on the soft constraints. Soft constraints can be added pointwise. The sum of two soft constraints is a soft constraint with scope that is the union of their scopes. The value of any assignment to the scope is the sum of the two functions on that assignment.

Example 4.30: Consider functions c_1 and c_2 in the previous example. c_1+c_2 is a function with scope $\{A,B,C\}$, given by

c_1+c_2:

c_1+c_2:

A	*B*	*C*	Cost
1	1	1	10
1	1	2	7
1	2	1	2
...	...	...	...

The second value is computed as follows:

(c_1+c_2) (A=1,B=1,C=2)

$=c_1(A=1,B=1)+c_2(B=1,C=2)$

=5+2

=7

One way to find the optimal assignment, corresponding to **generate and test**, is to compute the sum of the soft constraints and to choose an assignment with minimum value. Later we consider other, more efficient, algorithms for optimization.

Hard constraints can be modeled as having a cost of infinity for violating a constraint. As long as the cost of an assignment is finite, it does not violate a hard constraint. An alternative is to use a large number - larger than the sum of the soft constraints could be - as the cost of violating a hard constraint. Then optimization can be used to find a solution with the fewest number of violated hard constraints and, among those, one with the lowest cost.

Optimization problems have one difficulty that goes beyond constraint satisfaction problems. It is difficult to know whether an assignment is optimal. Whereas, for a CSP, an algorithm can check whether an assignment is a solution by just considering the assignment and the constraints, in optimization problems an algorithm can only determine if an assignment is optimal by comparing it to other assignments.

Many of the methods for solving hard constraints can be extended to optimization problems, as outlined in the following sections.

4.10.1 Systematic Methods for Optimization

Arc consistency can be generalized to optimization problems by allowing pruning of dominated assignments. Suppose $c_1,...,c_k$ are the soft constraints that involve X. Let soft constraint $c= c_1+...+c_k$. Suppose Y are the variables, other than X, that are involved in c. A value v for variable X is **strictly dominated** if, for all values y of Y, some value v' of X exists such that $c(X=v',Y=y) < c(X=v,Y=y)$. Pruning strictly dominated values does not remove an optimal solution. The pruning of domains can be done repeatedly, as in the GAC algorithm.

Weakly dominated has the same definition as strictly dominated, but with "less than" replaced by "less than or equal to." If only one solution is required, weakly dominated values can be pruned sequentially. Which weakly dominated values are removed may affect which optimal solution is found, but removing a weakly dominated value does not remove all optimal solutions. As with arc consistency for hard constraints, pruning (strictly or weakly) dominated values can greatly simplify the problem but does not, by itself, always solve the problem.

Domain splitting can be used to build a search tree. Domain splitting picks some variable X and considers each value of X. Assigning a value to X allows the constraints that involve X to be simplified and values for other variables to be pruned. In particular, pruning weakly dominated values means that, when there is only one variable left, a best value for that variable can be computed. Repeated domain splitting can build a search tree, as in Figure 4.1, but with values at the leaves. By assigning costs when they can be determined, search algorithms such as A^* or branch-and-bound can be used to find an optimal solution.

Domain splitting can be improved via two techniques. First, if, under a split on X, an assignment to another variable does not depend on the value of X, the computation for that variable can be shared among the subtrees for the values of X; the value can be computed once and cached. Second, if removing a set of variables would disconnect the constraint graph, then when those variables have been assigned values, the disconnected components can be solved independently.

Variable elimination is the dynamic programming variant of domain splitting. The variables are eliminated one at a time. A variable *X* is eliminated as follows. Let *R* be the set of constraints that involve *X*. *T* is a new constraint whose scope is the union of the scopes of the constraints in *R* and whose value is the sum of the values of *R*. Let $V=scope(T) \setminus \{X\}$. For each value of the variables in *V*, select a value of *X* that minimizes *T*, resulting in a new soft constraint, *N*, with scope *V*. The constraint *N* replaces the constraints in *R*. This results in a new problem, with fewer variables and a new set of constraints, which can be solved recursively. A solution, *S*, to the reduced problem is an assignment to the variables in *V*. Thus, *T(S)*, the constraint *T* under the assignment *S*, is a function of *X*. An optimal value for *X* is obtained by choosing a value that results in the minimum value of *T(S)*.

```
Procedure VE_SC(Vs,Fs)
    Inputs
        Vs: set of variables
        Fs: set of constraints Output
        an optimal assignment to Vs.
    if (Vs contains a single element or Fs contains a single constraint) then
        let F be the sum of the constraints in Fs
        return assignment with minimum value in F
    else
        select X∈Vs according to some elimination ordering
        R={F ∈Fs: F  involves X}
        let T be the sum of the constraints in R
        N ←min_X T
        S←VE_SC(Vs \ {X}, Fs \ R∪{N})
        X_opt←argmin_X T(S)
        return S ∪{X=X_opt}
```

Figure 4.11: Variable elimination for optimizing with soft constraints

---

Figure 4.11 gives pseudocode for the VE algorithm. The elimination ordering can be given a priori or can be computed on the fly, for example, using the elimination ordering heuristics discussed for CSP VE. It is possible to implement this without storing *T* and only by constructing an extensional representation of *N*.

Example 4.31: Consider Example 4.29. First consider eliminating *A*. It appears in only one constraint, $c_1(A,B)$. Eliminating *A* gives $c_6(B) = argmin_A\ c_1(A,B)$:

B	Cost
1	0
2	2
3	2

The constraint $c_1(A,B)$ is replaced by $c_6(B)$.

Suppose *B* is eliminated next. *B* appears in three constraints: $c_2(B,C)$, $c_3(B,D)$, and $c_6(B)$. These three constraints are added, giving $c_2(B,C) + c_3(B,D) + c_6(B)$:

B	*C*	*D*	Cost
1	1	1	8
1	1	2	5
	...		
2	1	1	4
2	1	2	4
	...		
3	1	1	6
3	1	2	8

The constraints c_2, c_3, and c_6 are replaced by $c_7(C,D)=min_B (c_2(B,C) + c_3(B,D) + c_6(B))$:

C	*D*	Cost
1	1	4
1	2	4
	...	

There are now three remaining constraints: $c_4(C,E)$, $C_5(D,E)$, and $c_7(C,D)$. These can be optimized recursively.

Suppose the recursive call returns the solution $C=1$, $D=2$, $E=2$. An optimal value for B is the value that gives the minimum in $c_2(B,C=1) + c_3(B,D=2) + c_6(B)$, which is $B=2$.

From $c_1(A,B)$, the value of A that minimizes $c_1(A,B=2)$ is $A=1$. Thus, an optimal solution is $A=1, B=2, C=1, D=2, E=2$.

The complexity of VE depends on the structure of the constraint graph, as it does with hard constraints. Sparse graphs may result in small intermediate constraints in VE algorithms, including *VE_SC*. Densely connected graphs result in large intermediate constraints.

4.10.2 Local Search for Optimization

Local search is directly applicable to optimization problems, using the objective function of the optimization problem as the evaluation function of the local search. The algorithm runs for a certain amount of time (perhaps including random restarts to explore other parts of the search space), always keeping the best assignment found thus far, and returning this as its answer.

Local search for optimization has one extra complication that does not arise with only hard constraints: it is difficult to determine whether a total assignment is the best possible solution. A **local minimum** is a total assignment that is at least as good, according to the optimality criterion, as any of its neighbors. A **global minimum** is a total assignment that is at least as good as all of the other total assignments. Without systematically searching the other assignments, the algorithm may not know whether the best assignment found so far is a global optimum or whether a better solution exists in a different part of the search space.

When solving constrained optimization problems, with both hard and soft constraints, there could be a trade-off between violating hard constraints and making the evaluation function worse. We typically do not treat a violated constraint as having a cost of infinity, because then the algorithms would not distinguish violating one hard constraint from violating many. It is sometimes good to allow hard constraints to be temporarily violated in the search.

4.10.2.1 Continuous Domains

For optimization where the domains are continuous, a local search becomes more complicated because it is not obvious what the neighbors of a total assignment are. This problem does not arise with hard constraint satisfaction problems because the constraints implicitly discretize the space.

For optimization, **gradient descent** can be used to find a minimum value, and **gradient ascent** can be used to find a maximum value. Gradient descent is like walking downhill and always taking a step in the direction that goes down the most. The general idea is that the neighbor of a total assignment is to step downhill in proportion to the slope of the evaluation function h. Thus, gradient descent takes steps in each direction proportional to the negative of the partial derivative in that direction.

In one dimension, if X is a real-valued variable with the current value of v, the next value should be

$v-\eta\times(dh/dX)(v),$

where

- η, the **step size**, is the constant of proportionality that determines how fast gradient descent approaches the minimum. If η is too large, the algorithm can overshoot the minimum; if η is too small, progress becomes very slow.
- $(d\, h)/(d\, X)$, the derivative of h with respect to X, is a function of X and is evaluated for $X=v$.

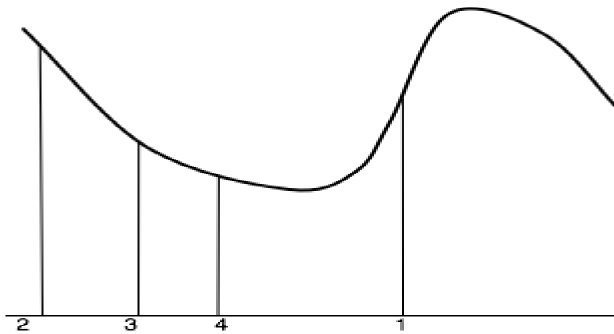


Figure 4.12: Gradient descent

Example 4.32: Figure 4.12 shows a typical one-dimensional example for finding a local minimum of a one-dimensional function. It starts at a position marked as 1. The derivative is a big positive value, so it takes a step to the left to position 2. Here the derivative is negative, and closer to zero, so it takes a smaller step to the right to position 3. At position 3, the derivative is negative and closer to zero, so it takes a smaller step to the right. As it approaches the local minimum value, the slope becomes closer to zero and it takes smaller steps.

For multidimensional optimization, when there are many variables, gradient descent takes a step in each dimension proportional to the partial derivative of that dimension. If $\langle X_1,..., X_n\rangle$ are the variables that have to be assigned values, a total assignment corresponds to a tuple of values $\langle v_1,..., v_n\rangle$. Assume that the evaluation function, *h*, is differentiable. The next neighbor of the total assignment $\langle v_1,..., v_n\rangle$ is obtained by moving in each direction in proportion to the slope of *h* in that direction. The new value for X_i is

$$v_i - \eta \times (\partial h / \partial X_i)(v_1,..., v_n),$$

where η is the **step size**. The partial derivative, $(\partial h / \partial X_i)$, is a function of $X_1,...,X_n$. Applying it to the point $(v_1,..., v_n)$ gives

$$(\partial h / \partial X_i)(v_1,..., v_n) = \lim_{\varepsilon \to \infty} (h(v_1,...,v_i+\varepsilon,..., v_n)- h(v_1,...,v_i,..., v_n))/\varepsilon.$$

If the partial derivative of *h* can be computed analytically, it is usually good to do so. If not, it can be estimated using a small value of ε.

Gradient descent is used for parameter learning, in which there may be thousands of real-valued parameters to be optimized. There are many variants of this algorithm that we do not discuss. For example, instead of using a constant step size, the algorithm could do a binary search to determine a locally optimal step size.

3.7 Genetic Algorithms

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. **They are commonly used to generate high-quality solutions for optimization problems and search problems.**

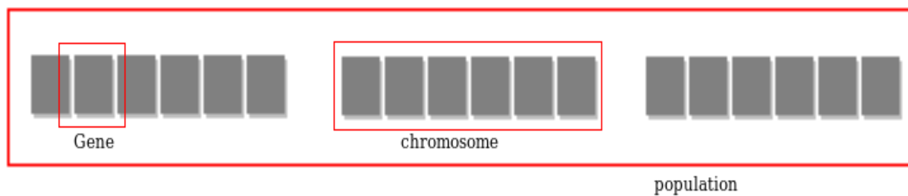
Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate "survival of the fittest" among individual of consecutive generation for solving a problem. **Each generation consist of a population of individuals** and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

Genetic algorithms are based on an analogy with genetic structure and behavior of chromosome of the population. Following is the foundation of GAs based on this analogy –

1. Individual in population compete for resources and mate
2. Those individuals who are successful (fittest) then mate to create more offspring than others
3. Genes from "fittest" parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.
4. Thus each successive generation is more suited for their environment.

Search space

The population of individuals are maintained within search space. Each individual represent a solution in search space for given problem. Each individual is coded as a finite length vector (analogous to chromosome) of components. These variable components are analogous to Genes. Thus a chromosome (individual) is composed of several genes (variable components).



Fitness Score

A Fitness Score is given to each individual which **shows the ability of an individual to "compete"**. The individual having optimal fitness score (or near optimal) are sought.

The GAs maintains the population of n individuals (chromosome/solutions) along with their fitness scores.The individuals having better fitness scores are given more chance to reproduce than others. The individuals with better fitness scores are selected who mate and produce **better offspring** by combining chromosomes of parents. The population size is static so the room has to be created for new arrivals. So, some individuals die and get replaced by new arrivals eventually creating new generation when all the mating opportunity of the old population is exhausted. It is hoped that over successive generations better solutions will arrive while least fit die.

Each new generation has on average more "better genes" than the individual (solution) of previous generations. Thus each new generations have better **"partial solutions"** than previous generations. Once the offsprings produced having no significant difference than offspring produced by previous populations, the population is converged. The algorithm is said to be converged to a set of solutions for the problem.

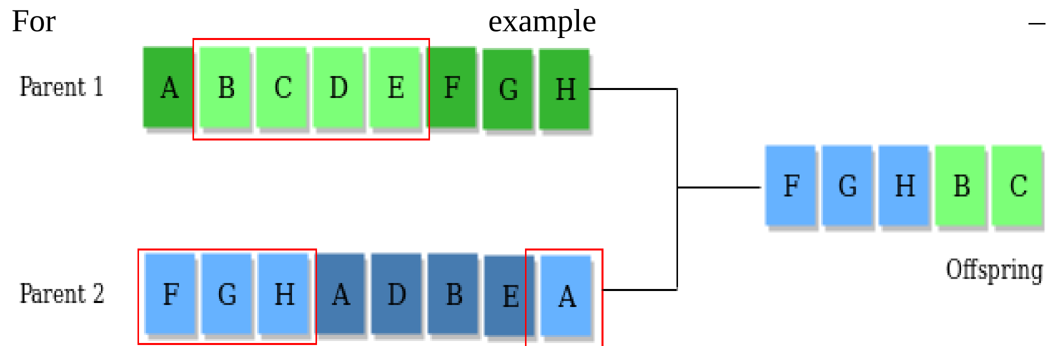
Operators of Genetic Algorithms

Once the initial generation is created, the algorithm evolve the generation using following operators –

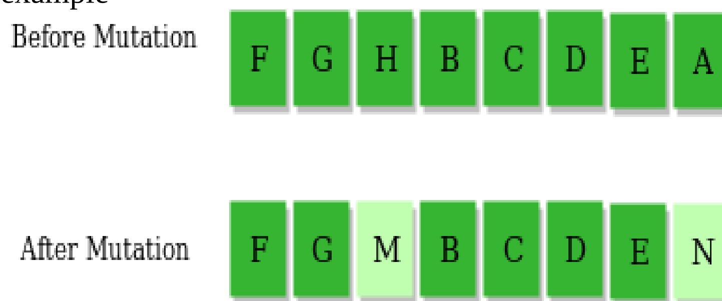
1) Selection Operator: The idea is to give preference to the individuals with good fitness scores and allow them to pass there genes to the successive generations.

2) Crossover Operator: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring).

For example –



3) Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in population to avoid the premature convergence. For example –



The whole algorithm can be summarized as –

1) Randomly initialize populations p
2) Determine fitness of population
3) Untill convergence repeat:
    a) Select parents from population
    b) Crossover and generate new population
    c) Perform mutation on new population
    d) Calculate fitness for new population

Example problem and solution using Genetic Algorithms

Given a target string, the goal is to produce target string starting from a random string of the same length. In the following implementation, following analogies are made –

- Characters A-Z, a-z, 0-9 and other special symbols are considered as genes
- A string generated by these character is considered as chromosome/solution/Individual

Why use Genetic Algorithms

- They are Robust
- Provide optimisation over large space state.
- Unlike traditional AI, they do not break on slight change in input or presence of noise

Application of Genetic Algorithms

Genetic algorithms have many applications, some of them are –

- Recurrent Neural Network
- Mutation testing
- Code breaking
- Filtering and signal processing
- Learning fuzzy rule base etc

Check your Progress
Note: a. Write your answer in the space given below
b. Compare your answer with those given at the end of the unit.

1. Explain Genetic Algorithm.

…………………………………………………………………………………

…………………………………………………………………………………

3.8 Terminologies

Here is the list of frequently used terms in the domain of AI −

Sr.No	Term & Meaning
1	**Agent** Agents are systems or software programs capable of autonomous, purposeful and reasoning directed towards one or more goals. They are also called assistants, brokers, bots, droids, intelligent agents, and software agents.
2	**Autonomous Robot** Robot free from external control or influence and able to control itself independently.
3	**Backward Chaining** Strategy of working backward for Reason/Cause of a problem.
4	**Blackboard** It is the memory inside computer, which is used for communication between the cooperating expert systems.
5	**Environment** It is the part of real or computational world inhabited by the agent.
6	**Forward Chaining** Strategy of working forward for conclusion/solution of a problem.
7	**Heuristics** It is the knowledge based on Trial-and-error, evaluations, and experimentation.
8	**Knowledge Engineering** Acquiring knowledge from human experts and other resources.
9	**Percepts** It is the format in which the agent obtains information about the environment.

10 **Pruning**

Overriding unnecessary and irrelevant considerations in AI systems.

11 **Rule**

It is a format of representing knowledge base in Expert System. It is in the form of IF-THEN-ELSE.

12 **Shell**

A shell is a software that helps in designing inference engine, knowledge base, and user interface of an expert system.

13 **Task**

It is the goal the agent is tries to accomplish.

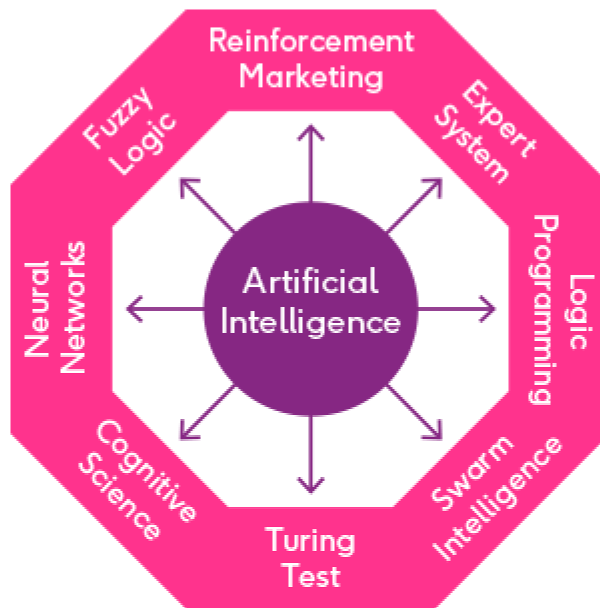
14 **Turing Test**

A test developed by Allan Turing to test the intelligence of a machine as compared to human intelligence.

We do not need to know everything in a conversation, but we should at least know the terms used in the conversation.

If we are talking about physics we need to know for example that when we are talking about ***velocity*** we mean the speed that an object takes to travel a space in a certain period of time. I think in Artificial Intelligence shouldn't be different, so, in this post, I'll let you know the meaning of the most used terminologies (and their acronym) so that the next time you meet with an AI post, you can read it with a deep understanding.

AI (*Artificial Intelligence*) — The first thing we need to do is understand what an AI actually is. The term "artificial intelligence" refers to a specific field of computer science that focuses on creating systems capable of gathering data and making decisions and/or solving problems.



Some parts of AI

AGI (*Artificial General Intelligence*) — is an emerging field aiming at the building of "thinking machines"; that is, general-purpose systems with intelligence comparable to that of the human mind, also called "Strong AI", "Human-level AI", etc.

ANI (*Artificial Narrow Intelligence*) — A one trick pony, they can play chess, recognize faces, translate foreign languages.

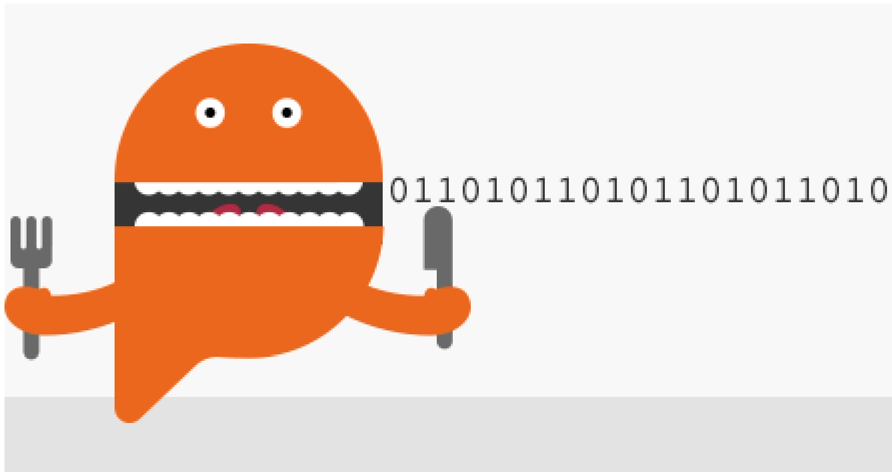
ASI (*Artificial Super Intelligence*) — Smarter than the best human brains and has the ability to apply that to absolutely anything (This is the AI that people like Stephen Hawking, Elon Musk, etc. are scared of).

Agent — also called assistants, brokers, bots, intelligent agents is an autonomous entity which observes through sensors and acts upon an environment using actuators.

Chatbot — A computer program that conducts conversations with human users by simulating how humans would behave as a conversational partner.

Data — Any collection of information converted into a digital form.

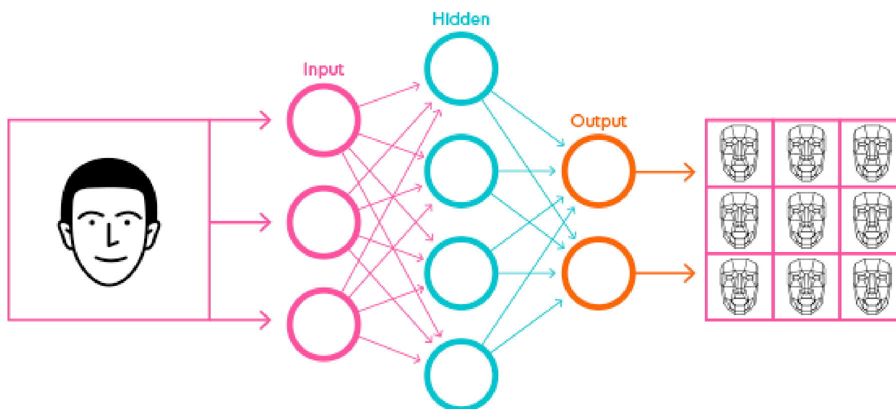
Data Mining — The process by which patterns are discovered within large sets of data with the goal of extracting useful information from it.



Data being processed

Data Mining — The process by which patterns are discovered within large sets of data with the goal of extracting useful information from it.

Deep Learning — A subset of AI and Machine learning in which Neural networks are “layered”, combined with plenty of computing power, and given a large measure of training data to create extremely powerful learning models capable of processing data in new and exciting ways in a number of areas, e.g. advancing the field of computer vision.



Neural Network

Genetic Algorithm — A method for solving optimization problems by mimicking the process of natural selection and biological evolution. The algorithm randomly selects pairs of individuals from the population (whereby the best performing individuals are more likely to be chosen) to be used as parents.

Heuristics — It is the knowledge based on Trial-and-error, evaluations, and experimentation.

ML (*Machine Learning*) — A subsetof AI in which computer programs and algorithms can be designed to “learn” how to complete a specified task, with increasing efficiency and effectiveness as it develops. Such programs can use past performance data to predict and improve future performance.

NLG (*Natural Language Generation*) — A machine learning task in which an algorithm attempts to generate language that is comprehensible and human-sounding. The end goal is to produce computer-generated language that is indiscernible from language generated by humans

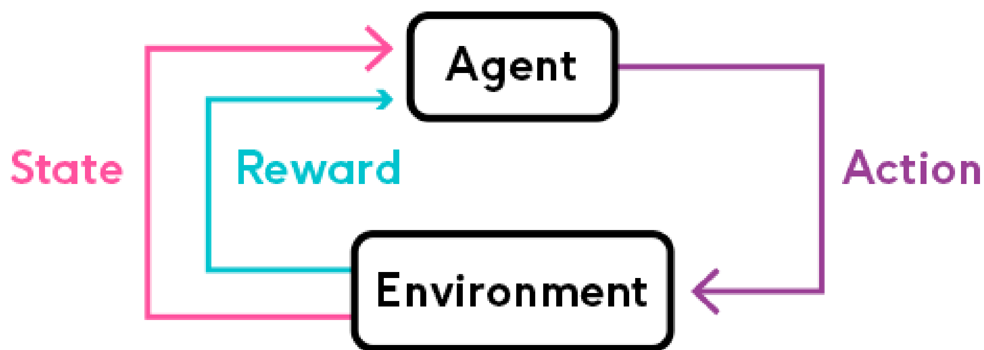
NLP (*Natural Language Processing*) — The ability of computers to understand, or process natural human languages and derive meaning from them. NLP typically involves machine interpretation of text or speech recognition
RNN (Recurrent Neural Network) — A type of artificial neural network in which recorded data and outcomes are fed back through the network forming a cycle.

OCR (*Optical Character Recognition*)— A computer system that takes images of typed, handwritten or printed text and converts them into machine-readable text.

Pruning — Overriding unnecessary and irrelevant considerations in AI systems.

RNN (*Recurrent Neural Network*) — A type of artificial neural network in which recorded data and outcomes are fed back through the network forming a cycle.

Reinforcement Learning — A type of machine learning in which machines are "taught" to achieve their target function through a process of experimentation and reward. In reinforcement learning, the machine receives positive reinforcement when its processes produce the desired result, and negative reinforcement when they do not.



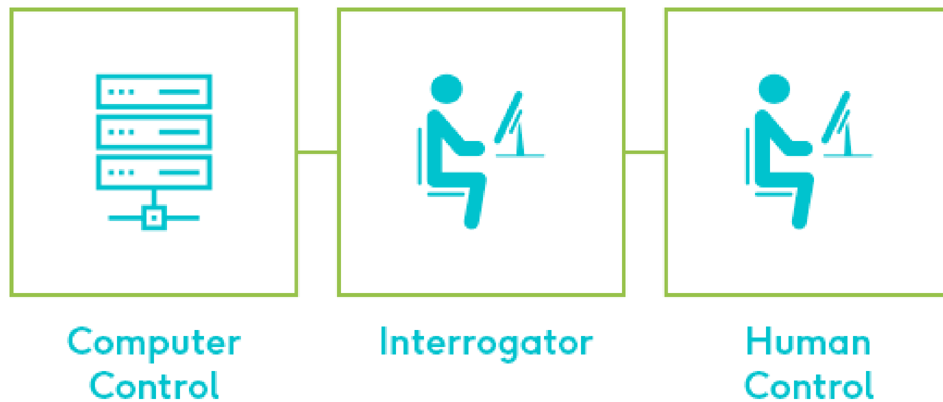
Reinforcement Learning

Rule — It is a format of representing knowledge base in Expert System. It is in the form of IF-THEN-ELSE

Supervised learning: A type of machine learning in which human input and supervision are an integral part of the machine learning process on an ongoing basis, like a teacher supervising a student; more common than unsupervised learning.

Strong AI — An area of AI development that is working toward the goal of making AI systems that are as useful and skilled as the human mind.

Turing Test — A test developed by Alan Turing 1950, which is meant as a means to identify true artificial intelligence. The test is based on a process in which a series of judges attempt to discern interactions with a control (human) from interactions with the machine (computer) being tested.



Turing Test

Unsupervised learning: A type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses.

As artificial intelligence becomes more complex, evolving technologies and the jargon associated with it might sound unfamiliar or strange to you. In this article, we have compiled the most important terms that are related to AI for you to flaunt in your next meeting.

Analogical Reasoning: The term analogical generally refers to non-digital data but when it comes to the field of AI, analogical reasoning is the process where people (scientists) draw conclusions based on past results. It is more like predicting stock markets.

Artificial Neuron Networks: Or connectionist systems is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data input.

Autonomic computing: Refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users.

Backpropagation: Is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network. It is commonly used to train deep neural networks, a term referring to neural networks with more than one hidden layer.

Backward chaining: It is used in automated theorem provers, inference engines, proof assistants, and other artificial intelligence applications.

Bayesian programming: Bayes' Theorem is the central concept behind this programming approach, which states that the probability of something occurring in the future can be inferred by past conditions related to the event

Behaviour informatics: (BI) is the informatics of behaviour analysis as to obtain behaviour intelligence and behaviour insights

Behaviour tree: A Behavior Tree (BT) is a mathematical model of plan execution used in computer science, robotics, control systems and video games. They describe switchings between a finite set of tasks in a modular fashion

Case-based reasoning(CBR): Broadly construed, is the process of solving new problems based on the solutions of similar past problems.

Data mining: is the process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems.

Decision boundary: In the case of backpropagation based artificial neural networks or perceptrons, the type of decision boundary that the network can learn is determined by the number of hidden layers the network has.

Decision tree learning: Uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves).

Evolutionary algorithm: Is a subset of evolutionary computation,[156] a generic population-based metaheuristic optimization algorithm. An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection.

Feature extraction: In machine learning, pattern recognition and in image processing, feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations.

Feature selection: In machine learning and statistics, feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction.

Forward chaining: Or forward reasoning is one of the two main methods of reasoning when using an inference engine and can be described logically as repeated application of modus ponens. Forward chaining is a popular implementation strategy for expert systems, business and production rule systems. The opposite of forwarding chaining is backward chaining.

Generative adversarial network (GAN): Is a class of machine learning systems. Two neural networks contest with each other in a zero-sum game framework.

Genetic algorithm (GA): Is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection

Graph database (GD): Is a database that uses graph structures for semantic queries with nodes, edges and properties to represent and store data.

Incremental learning: Is a method of machine learning, in which input data is continuously used to extend the existing model's knowledge

Named-entity recognition (NER): Also known as entity identification, entity chunking and entity extraction) is a subtask of information extraction that seeks to locate and classify named entity mentions in unstructured text into pre-defined categories such as the person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc.

Pattern recognition: Is the automated recognition of patterns and regularities in data. Pattern recognition is closely related to artificial intelligence and machine learning,[1] together with applications such as data mining and knowledge discovery in databases (KDD), and is often used interchangeably with these terms

Reinforcement learning (RL): Is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Reinforcement learning is considered as one of three machine learning paradigms, alongside supervised learning and unsupervised learning

Spatial-temporal reasoning: Iis an area of artificial intelligence which draws from the fields of computer science, cognitive science, and cognitive psychology. The theoretic goal—on the cognitive side—involves representing and reasoning spatial-temporal knowledge in mind.

Unsupervised learning: Is a term used for Hebbian learning, associated to learning without a teacher, also known as self-organization and a method of modelling the probability density of inputs.

Check your Progress

Note: a. Write your answer in the space given below

b. Compare your answer with those given at the end of the unit.

i. Define Feature selection.

………………………………………………………………………………………

………………………………………………………………………………………

3.9 Unit – End Exercise

1. Define Feature Selection
2. Explain Reinforcement Learning
3. Define Uniform Cost Search
4. Explain Genetic Algorithm

3.10 Answers to Check Your Progress

1. **Feature selection:** In machine learning and statistics, feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction.
2. **Reinforcement Learning** — A type of machine learning in which machines are "taught" to achieve their target function through a process of experimentation and reward. In reinforcement learning, the machine receives positive reinforcement when its processes produce the desired result, and negative reinforcement when they do not.
3. UCS is different from BFS and DFS because here the costs come into play. In other words, traversing via different edges might not have the same cost. The goal is to find a path where the cumulative sum of costs is least.
4. Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. **They are commonly used to generate high-quality solutions for optimization problems and search problems.**

3.11 Suggested Readings

1. https://www.geeksforgeeks.org/genetic-algorithms/
2. https://www.geeksforgeeks.org/search-algorithms-in-ai/
3. https://artint.info/html/ArtInt_83.html
4. https://guttulus.com/what-is-local-search-algorithm-in-artificial-intelligence/
5. https://artint.info/html/ArtInt_93.html
6. https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_quick_guide.htm
7. https://medium.com/machine-learning-world/artificial-intelligence-terminologies-260f1d6d299f

UNIT – IV Knowledge Representation

NOTES

Structure

4.1 Introduction

Knowledge is the body of facts and principles. Knowledge can be language, concepts, procedures, rules, ideas, abstractions, places, customs, and so on. Study of knowledge is called Epistemology. The main problem of modern Artificial Intelligence is Knowledge representation (KR), which means encoding real-world, 'common sense' knowledge in a format of both readable and understandable by the computer. Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. But how machines do all these things comes under knowledge representation and reasoning.

$$knows\ Whether\ (a,p) \Leftrightarrow knows\ (a,p) \lor knows\ (a,"\neg p")$$

Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.

- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

4.1.1 Knowledge and belief

The relation between believing and knowing has been studied extensively in philosophy. It is commonly said that knowledge is justified true belief. That is, if you believe something for an unassailably good reason, and if it is actually true, then you know it.

Let Knows (a,p) mean that agent a knows that proposition p is true. It is also possible to define other kinds of knowing. For example, here is a definition of "knowing whether":

Continuing our example, Lois knows whether Clark can fly if she either knows that Clark can fly or knows that he cannot.

The concept of "knowing what" is more complicated. One is tempted to say that an agent knows what Bob's phone number is if there is some x for which the agent knows Phone Number (Bob) = x. But that is not enough, because the agent might know that Alice and Bob have the same number (i.e., Phone Number {Bob) = Phone Number (Alice)), but if Alice's number is unknown, that isn't much help. A better definition of "knowing what" says that the agent has to be aware of some x that is a string of digits and that is Bob's number:

$$knows\ what\ (a."\text{phone number(b)}") \Leftrightarrow$$

$$\exists x\ knows\ (a,"x = ("\text{phone number(b)}") \land x \in DigitStrings$$

Of course, for other questions we have different criteria for what is an acceptable answer. For the question "what is the capital of New York," an acceptable answer is a proper name, "Albany," not something like "the city where the state house is." To handle this, we will make Knows what a three-place relation: it takes an agent, a string representing a term, and a category to which the answer must belong.

For example, we might have the following:

Knows What (Agent, "Capital {New York)", Proper Names).

Knows What {Agent, "Phone Number (Bob)'\ Digit Strings).

4.1.2 Knowledge, Time and Action

In most real situations, an agent will be dealing with beliefs—its own or those of other agents—that change over time. The agent will also have to make plans that involve changes to its own beliefs, such as buying a map to find out how to get to Bucharest. As with other predicates, we can reify Believes and talk about beliefs occurring over some period. For example, to say that Lois believes today that Superman can fly, we write

T (Believes (Lois, “Flies (Superman’), Today).

If the object of belief is a proposition that can change over time, then it too can be described using the T operator within the string. For example, Lois might believe today that that Superman could fly yesterday:

T (Believes (Lois, “T (Flies (Superman), Yesterday)”, Today). Given a way to describe beliefs over time, we can use the machinery of event calculus to make plans involving beliefs. Actions can have knowledge preconditions and knowledge effects. For example, the action of dialing a person’s number has the precondition of knowing the number, and the action of looking up the number has the effect of knowing the number. We can describe the latter action using the machinery of event calculus:

Initiates (Lookup (a, “Phone Number (6)”),

Knows What (a, “Phone Number (by’, Digit Strings), t)

Plans to gather and use information are often represented using a shorthand notation called runtime variables, which is closely related to the unquoted-variable convention described earlier. For example, the plan to look up Bob’s number and then dial it can be written as

[Lookup (Agent, “Phone Number (Bob)”, n), Dial (n)].

Here, n is a runtime variable whose value will be bound by the Lookup action and can then be used by the Dial action. Plans of this kind occur frequently in partially observable domains.

What to Represent:

Following are the kind of knowledge which needs to be represented in AI systems:

- Object: All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- Events: Events are the actions which occur in our world.
- Performance: It describes behavior which involves knowledge about how to do things.
- Meta-knowledge: It is knowledge about what we know.
- Facts: Facts are the truths about the real world and what we represent.
- Knowledge-Base: The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Check your Progress-1

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Knowledge.

………………………………………………………………………………..

………………………………………………………………………………..

……………………………………………………………………………….

………………………………………………………………………………

……………………………………………………………………………….

4.2 Types of knowledge

The types of knowledge include procedural knowledge, declarative knowledge and heuristic knowledge.

4.2.1 Procedural knowledge

Procedural knowledge is compiled or processed form of information. Procedural knowledge is related to the performance of some task. It is a representation in which the control information, to use the knowledge, is embedded in the knowledge itself.

E.g. computer programs, directions, and recipes; these indicate specific use or implementation.

For example, sequence of steps to solve a problem is procedural knowledge.

Here, the knowledge is a mapping process between domains that specify **"what to do when"** and the representation is of **"how to make it"** rather than *"what it is"*. The procedural knowledge: may have inferential efficiency, but no inferential adequacy and acquisitional efficiency. These are represented as small programs that know how to do specific things, how to proceed.

Example: A parser in a natural language has the knowledge that a noun phrase may contain articles, adjectives and nouns. It thus accordingly calls routines that know how to process articles, adjectives, nouns.

4.2.2 Declarative knowledge

Declarative knowledge is passive knowledge in the form of statements of facts about the world. For example, mark statement of a student is declarative knowledge. It is a statement in which knowledge is specified, but the use to which that knowledge is to be put is not given.

E.g. laws, people's name; these are facts which can stand alone, not dependent on other knowledge.
Here, the knowledge is based on declarative facts about *axioms* and *domains.*

Axioms are assumed to be true unless a counter example is found to invalidate them.

− Domains represent the physical world and the functionality.

4.2.3 Heuristic knowledge

Heuristics knowledge is rules of thumb or tricks. Heuristic knowledge is used to make judgments and also to simplify solution of problems. It is acquired through experience. An expert uses his knowledge that he has gathered due to his experience and learning. Heuristic knowledge is representing knowledge of some experts in a field or subject. Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

4.2.4 Relational Knowledge:

This knowledge associates elements of one domain with another domain.

Relational knowledge is made up of objects consisting of attributes and their corresponding associated values. The result of this knowledge type is a mapping of elements among different domains.

The table below shows a simple way to store facts.

The facts about a set of objects are put systematically in columns. − This representation provides little opportunity for inference.

Simple Relational Knowledge

Player	Height	Weight	Bats - Throws
Aaron	6-0	180	Right - Right
Mays	5-10	170	Right - Right
Ruth	6-2	215	Left - Left
Williams	6-3	205	Left - Right

Given the facts it is not possible to answer simple question such as:

Who is the heaviest player? ".

But if a procedure for finding heaviest player is provided, then these facts will enable that procedure to compute an answer.

We can ask things like who "bats – left" and "throws – right".

The relation between knowledge and intelligence:

Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behavior in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

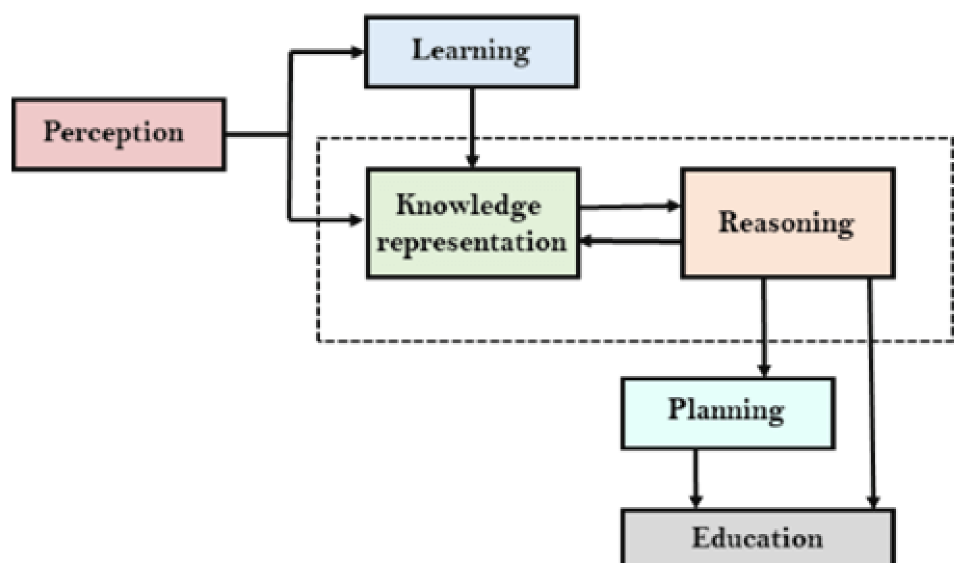
Let's suppose if you met some person who is speaking in a language which you don't know, then how you will able to act on that. The same thing applies to the intelligent behavior of the agents.

As we can see in below diagram, there is one decision maker which act by sensing the environment and using knowledge. But if the knowledge part will not present then, it cannot display intelligent behavior.

AI knowledge cycle:

An Artificial intelligence system has the following components for displaying intelligent behavior:

- Perception
- Learning
- Knowledge Representation and Reasoning
- Planning
- Execution



The above diagram is showing how an AI system can interact with the real world and what components help it to show intelligence. AI system has Perception component by which it retrieves information from its environment. It can be visual, audio or another form of sensory input. The learning component is responsible for learning from data captured by Perception comportment. In the complete cycle, the main components are knowledge representation and Reasoning. These two components are involved in showing the intelligence in machine-like humans. These two components are independent with each other but also coupled together. The planning and execution depend on analysis of Knowledge representation and reasoning.

Check your Progress-2

Note: a. Write your answer in the space given below.

ii. Define Procedural Knowledge.

…………………………………………………………………………

………………………………………………………………………..

………………………………………………………………………..

iii. Write your understanding about Heuristics knowledge

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

4.3 Approaches to knowledge representation:

There are mainly four approaches to knowledge representation, which are given below:

4.3.1. Simple relational knowledge:

- ❖ It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- ❖ This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- ❖ This approach has little opportunity for inference.

Example: The following is the simple relational knowledge representation.

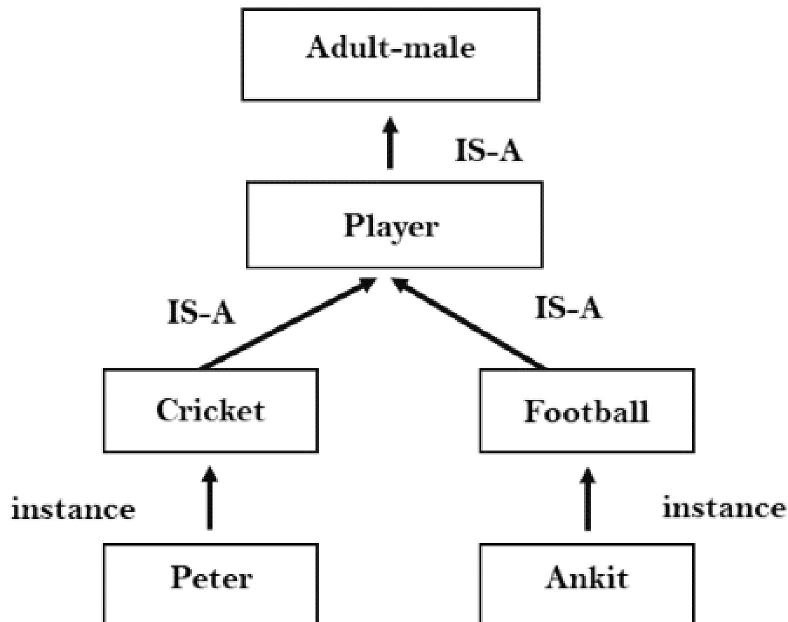
Player	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

4.3.2. Inheritable knowledge:

In the inheritable knowledge approach, all data must be stored into a hierarchy of classes. All classes should be arranged in a generalized form or a hierarchal manner. In this approach, we apply inheritance property.

- o Elements inherit values from other members of a class.
- o This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- o Every individual frame can represent the collection of attributes and its value.
- o In this approach, objects and values are represented in Boxed nodes.
- o We use Arrows which point from objects to their values.

Example:



4.3.3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.

Example: Let's suppose there are two statements:

a) Marcus is a man
b) All men are mortal

Then it can represent as;
man(Marcus)
∀x = man (x) ----------> mortal (x)s

4.3.4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.

- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

Requirements for knowledge Representation system:

A good knowledge representation system must possess the following properties.

i. **Representational Accuracy:**
KR system should have the ability to represent all kind of required knowledge.

ii. **Inferential Adequacy:**
KR system should have ability to manipulate the representational structures to produce new knowledge corresponding to existing structure.

iii. **Inferential Efficiency:**
The ability to direct the inferential knowledge mechanism into the most productive directions by storing appropriate guides.

iv. **Acquisitional efficiency-** The ability to acquire the new knowledge easily using automatic methods.

Check your Progress-3

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the chapter.

iv. Write the approaches to knowledge representation

…………………………………………………………………………..

…………………………………………………………………………..

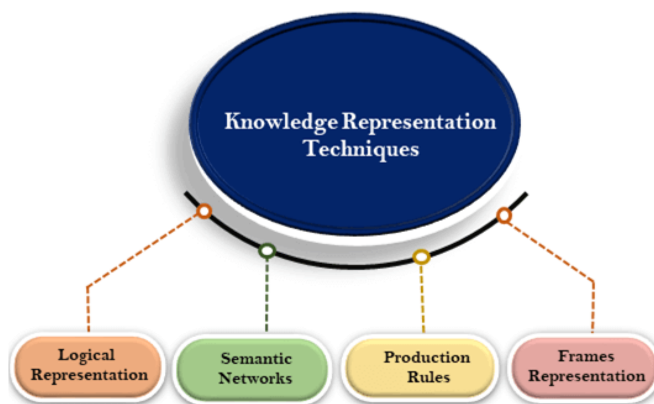
…………………………………………………………………………..

…………………………………………………………………………..

4.4 Techniques of Knowledge Representation

There are mainly four ways of knowledge representation which are given as follows:

1. Logical Representation
2. Semantic Network Representation
3. Frame Representation
4. Production Rules



4.4.1. Logical Representation

Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation. Logical representation means drawing a conclusion based on various conditions. This representation lays down some important communication rules. It consists of precisely defined syntax and semantics which supports the sound inference. Each sentence can be translated into logics using syntax and semantics.

Syntax:

- Syntaxes are the rules which decide how we can construct legal sentences in the logic.
- It determines which symbol we can use in knowledge representation.
- How to write those symbols.

Semantics:

- Semantics are the rules by which we can interpret the sentence in the logic.
- Semantic also involves assigning a meaning to each sentence.

Logical representation can be categorised into mainly two logics:

a. Propositional Logics

b. Predicate logics

Advantages of logical representation:

- Logical representation enables us to do logical reasoning.
- Logical representation is the basis for the programming languages.

Disadvantages of logical Representation:

- Logical representations have some restrictions and are challenging to work with.
- Logical representation technique may not be very natural, and inference may not be so efficient.

4.4.2 Semantic Network Representation

Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Semantic networks can categorize the object in different forms and can also link those objects. Semantic networks are easy to understand and can be easily extended.

This representation consists of mainly two types of relations:

a. IS-A relation (Inheritance)

b. Kind-of-relation

Example: Following are some statements which we need to represent in the form of nodes and arcs.

Statements:

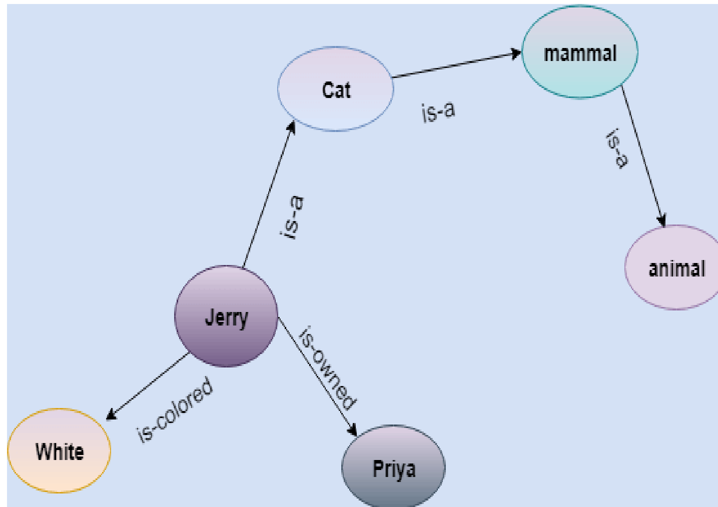
a. Jerry is a cat.

b. Jerry is a mammal

c. Jerry is owned by Priya.

d. Jerry is brown colored.

e. All Mammals are animal.



In the above diagram, we have represented the different type of knowledge in the form of nodes and arcs. Each object is connected with another object by some relation.

Drawbacks in Semantic representation:

- ✓ Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions. It might be possible in the worst case scenario that after traversing the entire tree, we find that the solution does not exist in this network.
- ✓ Semantic networks try to model human-like memory (Which has 1015 neurons and links) to store the information, but in practice, it is not possible to build such a vast semantic network.
- ✓ These types of representations are inadequate as they do not have any equivalent quantifier, e.g., for all, for some, none, etc.
- ✓ Semantic networks do not have any standard definition for the link names.
- ✓ These networks are not intelligent and depend on the creator of the system.

Advantages of Semantic network:

- ✓ Semantic networks are a natural representation of knowledge.
- ✓ Semantic networks convey meaning in a transparent manner.
- ✓ These networks are simple and easily understandable.

4.4.3. Frame Representation

A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations. It consists of a collection of slots and slot values. These slots may be of any type and sizes. Slots have names and values which are called facets.

Facets: The various aspects of a slot is known as **Facets**. Facets are features of frames which enable us to put constraints on the frames. Example: IF-NEEDED facts are called when data of any particular slot is needed. A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values. A frame is also known as **slot-filter knowledge representation** in artificial intelligence.

Frames are derived from semantic networks and later evolved into our modern-day classes and objects. A single frame is not much useful. Frames system consists of a collection of frames which are connected. In the frame, knowledge about an object or event can be stored together in the knowledge base. The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions.

Example: 1

Let's take an example of a frame for a book

Slots	Filters
Title	Artificial Intelligence
Genre	Computer Science
Author	Peter Norvig
Edition	Third Edition
Year	1996
Page	1152

Example 2:

Let's suppose we are taking an entity, Peter. Peter is an engineer as a profession, and his age is 25, he lives in city London, and the country is England. So following is the frame representation for this:

Slots	Filter
Name	Peter
Profession	Doctor
Age	25
Marital status	Single
Weight	78

Advantages of frame representation:

- The frame knowledge representation makes the programming easier by grouping the related data.
- The frame representation is comparably flexible and used by many applications in AI.
- It is very easy to add slots for new attribute and relations.
- It is easy to include default data and to search for missing values.
- Frame representation is easy to understand and visualize.

Disadvantages of frame representation:

- In frame system inference mechanism is not be easily processed.
- Inference mechanism cannot be smoothly proceeded by frame representation.
- Frame representation has a much generalized approach.

4.4.4. Production Rules

Production rules system consist of (**condition, action**) pairs which mean, "If condition then action".

It has mainly three parts:

- The set of production rules
- Working Memory
- The recognize-act-cycle

In production rules agent checks for the condition and if the condition exists then production rule fires and corresponding action is carried out. The condition part of the rule determines which rule may be applied to a problem. And the action part carries out the associated problem-solving steps. This complete process is called a recognize-act cycle.

The working memory contains the description of the current state of problems-solving and rule can write knowledge to the working memory. This knowledge match and may fire other rules.

If there is a new situation (state) generates, then multiple production rules will be fired together, this is called conflict set. In this situation, the agent needs to select a rule from these sets, and it is called a conflict resolution.

Example:

- **IF (at bus stop AND bus arrives) THEN action (get into the bus)**
- **IF (on the bus AND paid AND empty seat) THEN action (sit down).**
- **IF (on bus AND unpaid) THEN action (pay charges).**
- **IF (bus arrives at destination) THEN action (get down from the bus).**

Advantages of Production rule:

1. The production rules are expressed in natural language.
2. The production rules are highly modular, so we can easily remove, add or modify an individual rule.

Disadvantages of Production rule:

1. Production rule system does not exhibit any learning capabilities, as it does not store the result of the problem for the future uses.
2. During the execution of the program, many rules may be active hence rule-based production systems are inefficient.

Check your Progress-4

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the chapter.

v. Write the techniques of knowledge representation

……………………………………………………………………………………..

……………………………………………………………………………………..

……………………………………………………………………………………..

vi. Write drawbacks in Semantic Network

……………………………………………………………………………………..

………………………………………………………………………………………

………………………………………………………………………………………

vii. State the three main parts of production rules

……………………………………………………………………………………..

……………………………………………………………………………………..

……………………………………………………………………………………..

4.5 Propositional Logic

Propositional logic

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.

Example:

a) It is Sunday.
b) The Sun rises from West (False proposition)
c) 3+3= 7(False proposition)
d) 5 is a prime number.

Following are some basic facts about propositional logic:

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.
- Statements which are questions, commands, or opinions are not propositions such as "**Where is Rohini**", "**How are you**", "**What is your name**", are not propositions.

Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

a. **Atomic Propositions**

b. **Compound propositions**

Atomic Proposition: Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Example:

a) 2+2 is 4, it is an atomic proposition as it is a **true** fact.
b) "The Sun is cold" is also a proposition as it is a **false** fact.

Compound proposition: Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

Example:

a) "It is raining today, and street is wet."
b) "Ankit is a doctor, and his clinic is in Mumbai."

4.5.1. Logical Connectives:

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

1. **Negation:** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
2. **Conjunction:** A sentence which has ∧ connective such as, **P ∧ Q** is called a conjunction.
   Example: Rohan is intelligent and hardworking. It can be written as,
   P= Rohan is intelligent,
   Q= Rohan is hardworking. → P∧ Q.

3. **Disjunction:** A sentence which has ∨ connective, such as **P ∨ Q**. is called disjunction, where P and Q are the propositions.
   Example: "Ritika is a doctor or Engineer",
   Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as **P ∨ Q**.
4. **Implication:** A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as
   If it is raining, then the street is wet.
   Let P= It is raining, and Q= Street is wet, so it is represented as P → Q
5. **Biconditional:** A sentence such as **P⇔ Q is a Biconditional sentence, example If I am breathing, then I am alive**
   P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

Following is the summarized table for Propositional Logic Connectives:

Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬ A or ¬ B

Truth Table:

In propositional logic, we need to know the truth values of propositions in all possible scenarios. We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called **Truth table**. Following are the truth table for all logical connectives:

For Negation:

P	¬P
True	**False**
False	**True**

For Conjunction:

P	Q	$P \wedge Q$
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

For disjunction:

P	Q	$P \vee Q$.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

For Implication:

P	Q	$P \rightarrow Q$
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**

For Biconditional:

P	Q	$P \Leftrightarrow Q$
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**True**

Truth table with three propositions:

We can build a proposition composing three propositions P, Q, and R. This truth table is made-up of 8n Tuples as we have taken three proposition symbols.

P	Q	R	¬R	P∨Q	P∨Q→¬R
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

4.5.2. Precedence of connectives:

Just like arithmetic operators, there is a precedence order for propositional connectors or logical operators. This order should be followed while evaluating a propositional problem. Following is the list of the precedence order for operators:

Precedence	**Operators**
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

4.5.3. Logical equivalence:

Logical equivalence is one of the features of propositional logic. Two propositions are said to be logically equivalent if and only if the columns in the truth table are identical to each other.

Let's take two propositions A and B, so for logical equivalence, we can write it as A⇔B. In below truth table we can see that column for ¬A∨ B and A→B, are identical hence A is Equivalent to B

A	B	¬A	¬A∨ B	A→B
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties of Operators:

- **Commutativity:**
  - P∧ Q= Q ∧ P, or
  - P ∨ Q = Q ∨ P.
- **Associativity:**
  - (P ∧ Q) ∧ R= P ∧ (Q ∧ R),
  - (P ∨ Q) ∨ R= P ∨ (Q ∨ R)
- **Identity element:**
  - P ∧ True = P,
  - P ∨ True= True.
- **Distributive:**
  - P∧ (Q ∨ R) = (P ∧ Q) ∨ (P ∧ R).
  - P ∨ (Q ∧ R) = (P ∨ Q) ∧ (P ∨ R).
- **DE Morgan's Law:**
  - ¬ (P ∧ Q) = (¬P) ∨ (¬Q)
  - ¬ (P ∨ Q) = (¬ P) ∧ (¬Q).
- **Double-negation elimination:**
  - ¬ (¬P) = P.

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic.
- Example:
    a. **All the girls are intelligent.**
    b. **Some apples are sweet.**
- Propositional logic has limited expressive power.
- In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

Check your Progress-5

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the chapter.

viii. Define Propositional Logic

…...

…...

…...

ix. Write about Logical equivalence.

…...

…...

…...

…...

4.6 Inference Rules

Inference:

In artificial intelligence, we need intelligent computers which can create new logic from old logic or by evidence, **so generating the conclusions from evidence and facts is termed as Inference**.

Inference rules:

Inference rules are the templates for generating valid arguments. Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.

In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:

Implication: It is one of the logical connectives which can be represented as P → Q. It is a Boolean expression.

Converse: The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as Q → P.

Contrapositive: The negation of converse is termed as contrapositive, and it can be represented as ¬ Q → ¬ P.

Inverse: The negation of implication is called inverse. It can be represented as ¬ P → ¬ Q.

From the above term some of the compound statements are equivalent to each other, which we can prove using truth table:

P	Q	P → Q	Q→ P	¬ Q → ¬ P	¬ P → ¬ Q.
T	T	T	T	T	T
T	F	F	T	F	T
F	T	T	F	T	F
F	F	T	T	T	T

Hence from the above truth table, we can prove that P → Q is equivalent to ¬ Q → ¬ P, and Q→ P is equivalent to ¬ P → ¬ Q.

4.6.1. Types of Inference rules:

4.6.1.1. Modus Ponens:

The Modus Ponens rule is one of the most important rules of inference, and it states that if P and P → Q is true, then we can infer that Q will be true. It can be represented as:

Notation for Modus ponens: $\frac{P \to Q,\ \ P}{\therefore Q}$

Example:

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
Statement-2: "I am sleepy" ==> P
Conclusion: "I go to bed." ==> Q.
Hence, we can say that, if P→ Q is true and P is true then Q will be true.

Proof by Truth table:

P	Q	$P \to Q$
0	0	0
0	1	1
1	0	0
1	1	1 ←

4.6.1.2. Modus Tollens:

The Modus Tollens rule state that if P→ Q is true and ¬ **Q is true, then** ¬ **P** will also true. It can be represented as:

Notation for Modus Tollens: $\frac{P \to Q,\ \sim Q}{\sim P}$

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
Statement-2: "I do not go to the bed."==> ~Q
Statement-3: Which infers that "**I am not sleepy**" => ~P

Proof by Truth table:

P	Q	$\sim P$	$\sim Q$	$P \to Q$
0	0	1	1	1 ←
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1

4.6.1.3. Hypothetical Syllogism:

The Hypothetical Syllogism rule state that if P→R is true whenever P→Q is true, and Q→R is true. It can be represented as the following notation:

Example:

Statement-1: If you have my home key then you can unlock my home. **P→Q**
Statement-2: If you can unlock my home then you can take my money. **Q→R**
Conclusion: If you have my home key then you can take my money. **P→R**

<u>Proof by truth table:</u>

P	Q	R	$P \to Q$	$Q \to R$	$P \to R$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1

4.6.1.4. Disjunctive Syllogism:

The Disjunctive syllogism rule state that if P∨Q is true, and ¬P is true, then Q will be true. It can be represented as:

Notation of Disjunctive syllogism: $\frac{P \lor Q,\ \neg P}{Q}$

Example:

Statement-1: Today is Sunday or Monday. ==>P∨Q
Statement-2: Today is not Sunday. ==> ¬P
Conclusion: Today is Monday. ==> Q

Proof by truth-table:

P	Q	¬P	P ∨ Q
0	0	1	0
0	1	1	1 ←
1	0	0	1
1	1	0	1

4.6.1.5. Addition:

The Addition rule is one the common inference rule, and it states that If P is true, then P∨Q will be true.

Notation of Addition: $\frac{P}{P \vee Q}$

Example:

Statement: I have a vanilla ice-cream. ==> P
Statement-2: I have Chocolate ice-cream.
Conclusion: I have vanilla or chocolate ice-cream. ==> (P∨Q)

Proof by Truth-Table:

P	Q	P ∨ Q
0	0	0
1	0	1 ←
0	1	1
1	1	1 ←

4.6.1.6. Simplification:

The simplification rule state that if **P∧ Q** is true, then **Q or P** will also be true. It can be represented as:

Notation of Simplification rule: $\frac{P \wedge Q}{Q}$ Or $\frac{P \wedge Q}{P}$

Proof by Truth-Table:

P	Q	P ∧ Q
0	0	0
1	0	0
0	1	0
1	1	1 ←

4.6.1.7. Resolution:

The Resolution rule state that if P∨Q and ¬ P∧R is true, then Q∨R will also be true. **It can be represented as**

Notation of Resolution $\frac{P \lor Q, \ \neg P \land R}{Q \lor R}$

Proof by Truth-Table:

P	¬P	Q	R	$P \lor Q$	¬P∧R	$Q \lor R$
0	1	0	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	0	0
1	0	0	1	1	0	1
1	0	1	0	1	0	1
1	0	1	1	1	0	1

Check your Progress-6

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the chapter.

x. Define Inference Rules

……………………………………………………………………………………..

……………………………………………………………………………………..

……………………………………………………………………………………..

xi. Write about Disjunctive syllogism rule.

……………………………………………………………………………………..

………………………………………………………………………………………

………………………………………………………………………………………

……………………………………………………………………………………..

4.7 Inference Methods

First-Order Logic in Artificial intelligence

In the topic of Propositional logic, we have seen that how to represent statements using propositional logic. But unfortunately, in propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

- **"Some humans are intelligent", or**
- **"Sachin likes cricket."**

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

First-Order logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:

- **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, ...
- **Relations: It can be unary relation such as:** red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has color, comes between
- **Function:** Father of, best friend, third inning of, end of, ...

As a natural language, first-order logic also has two main parts:

- **Syntax**
- **Semantics**

Syntax of First-Order logic:

The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as **Predicate (term1, term2, ..., term n)**.

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).

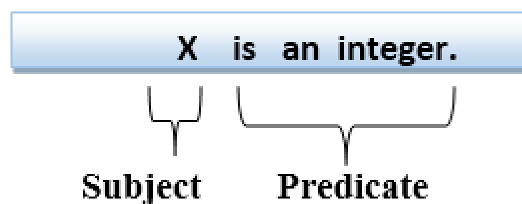
Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- **Subject:** Subject is the main part of the statement.
- **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression.

- There are two types of quantifier:
    a. **Universal Quantifier, (for all, everyone, everything)**
    b. **Existential quantifier, (for some, at least one).**

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

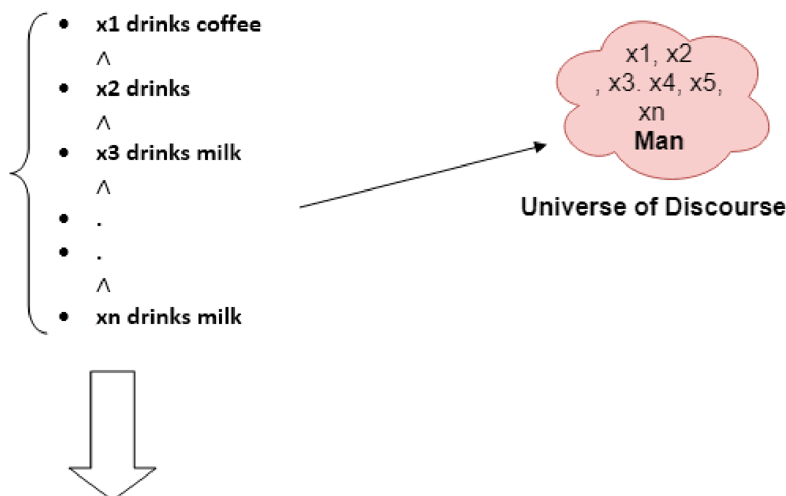
If x is a variable, then ∀x is read as:

- **For all x**
- **For each x**
- **For every x.**

Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:



∀x man(x) → drink (x, coffee).

It will be read as: There are all x where x is a man who drink coffee.

Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

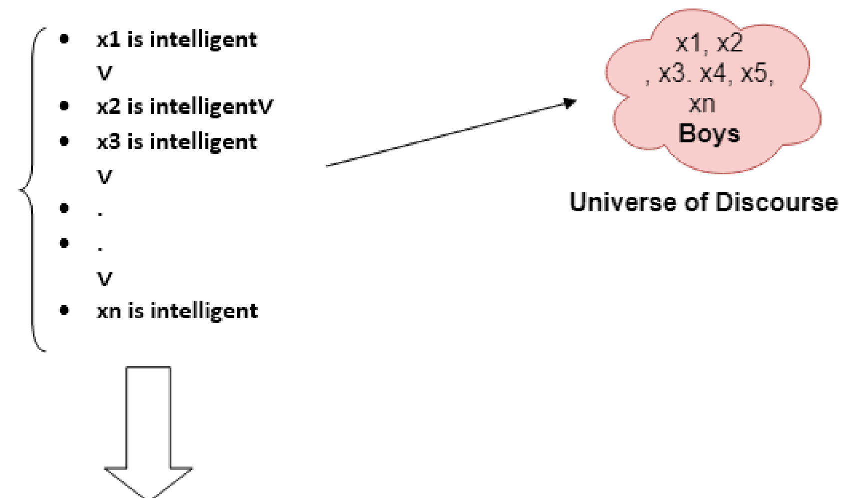
It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

- **There exists a 'x.'**
- **For some 'x.'**
- **For at least one 'x.'**

Example:

Some boys are intelligent.



∃x: boys(x) ∧ intelligent(x)

It will be read as: There are some x where x is a boy who is intelligent.

Points to remember:

- The main connective for universal quantifier **∀** is implication →.
- The main connective for existential quantifier **∃** is and **∧**.

Properties of Quantifiers:

- In universal quantifier, ∀x∀y is similar to ∀y∀x.
- In Existential quantifier, ∃x∃y is similar to ∃y∃x.
- ∃x∀y is not similar to ∀y∃x.

Some Examples of FOL using quantifier:

1. All birds fly.
In this question the predicate is "**fly(bird)**."
And since there are all birds who fly so it will be represented as follows.

∀x bird(x) →fly(x).

2. Every man respects his parent.
In this question, the predicate is "**respect(x, y)," where x=man, and y= parent**.
Since there is every man so will use ∀, and it will be represented as follows:

∀x man(x) → respects (x, parent).

3. Some boys play cricket.
In this question, the predicate is "**play(x, y)**," where x= boys, and y= game.

Since there are some boys so we will use **∃, and it will be represented as**:

∃x boys(x) → play(x, cricket).

4. Not all students like both Mathematics and Science.
In this question, the predicate is "**like(x, y)," where x= student, and y= subject**.
Since there are not all students, so we will use ∀ with negation, so following representation for this:

¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

5. Only one student failed in Mathematics.
In this question, the predicate is "**failed(x, y)," where x= student, and y= subject**.
Since there is only one student who failed in Mathematics, so we will use following representation for this:

∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

Free and Bound Variables:

The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic which are given below:

Free Variable: A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: ∀x ∃(y)[P (x, y, z)], where z is a free variable.

Bound Variable: A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

Example: ∀x [A (x) B(y)], here x and y are the bound variables.

Check your Progress-7

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the chapter.

i. Define Atomic Sentences.

…………………………………………………………………

ii. Write about Existential quantifiers.

…………………………………………………………………

End Unit Exercise:

1. Define Knowledge.
2. Differentiate between Knowledge and Belief.
3. Write a short note on Knowledge, Time and Action.
4. Explain the types of Knowledge.
5. Briefly discuss about the AI Knowledge Cycle.
6. Describe about an Approaches of Knowledge Representation.
7. Differentiate between Inheritable Knowledge and Inferential Knowledge.
8. Briefly discuss about the Techniques of Knowledge Representation.
9. Explain Propositional Logic.
10. Describe Inference Rules.
11. Explain the Methods of Inference.

Answers to check your progress:

i. Knowledge is the body of facts and principles. Knowledge can be language, concepts, procedures, rules, ideas, abstractions, places, customs, and so on.

ii. Procedural knowledge is related to the performance of some task. It is a representation in which the control information, to use the knowledge, is embedded in the knowledge itself.

iii. Heuristic knowledge is used to make judgments and also to simplify solution of problems.

iv. Simple Relational Knowledge, Inheritable Knowledge, Inferential Knowledge, Procedural Knowledge.

v. Logical, Semantic Network, Frame, Production Rules.

vi. Semantic networks take more computational time, do not have any Standard definition, not Intelligent and depend on creator.

vii. Set of Production Rules, Working Memory, and Recognize-act-cycle.

viii. Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false.

ix. Logical equivalence is one of the features of propositional logic. Two propositions are said to be logically equivalent if and only if the columns in the truth table are identical to each other.

x. Inference rules are the templates for generating valid arguments. Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.

xi. The Disjunctive syllogism rule state that if $P \lor Q$ is true, and $\neg P$ is true, then Q will be true.

xii. Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms. We can represent atomic sentences as **Predicate (term1, term2, ..., term n)**.

xiii. Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something. It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

Suggested Reading:

1. Stuart Russel, Peter Norvig, "Artificial Intelligence: A Modern Approach – 3 /e", 2014. Pearson Education.
2. Stuart Russel, Peter Norvig, "Artificial Intelligence: A Modern Approach – 2 /e", 2003. Pearson Education.
3. Elaine Rich, Kevin Knight, "Artificial Intelligence" 2/e, 1991, TMH.
4. Dan W. Patterson, "Introduction to Artificial Intelligence & Expert Systems", Seventh Indian Reprint 1999, EEE, PHI.

UNIT – V KNOWLEGDE ENGINEERING PROCESS

NOTES

Structure

5.1 Knowledge Engineering Process

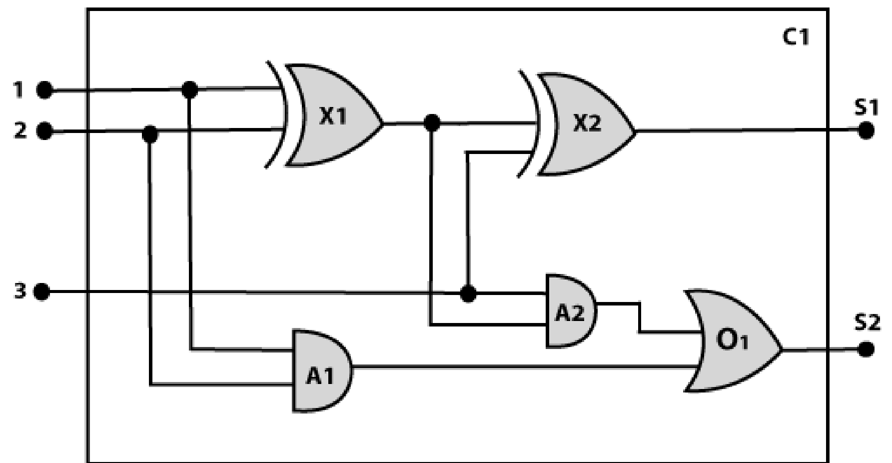
5.1.1 What is knowledge-engineering?

The process of constructing a knowledge-base in first-order logic is called as knowledge- engineering. In **knowledge-engineering**, someone who investigates a particular domain, learns important concept of that domain, and generates a formal representation of the objects, is known as **knowledge engineer**.

In this topic, we will understand the Knowledge engineering process in an electronic circuit domain, which is already familiar. This approach is mainly suitable for creating **special-purpose knowledge base**.

5.1.2. The process of knowledge-engineering:

Following are some main steps of the knowledge-engineering process. Using these steps, we will develop a knowledge base which will allow us to reason about digital circuit (**One-bit full adder**) which is given below



5.1.2.1. Identify the task:

The first step of the process is to identify the task, and for the digital circuit, there are various reasoning tasks.

At the first level or highest level, we will examine the functionality of the circuit:

- **Does the circuit add properly?**
- **What will be the output of gate A2, if all the inputs are high?**

At the second level, we will examine the circuit structure details such as:

- **Which gate is connected to the first input terminal?**
- **Does the circuit have feedback loops?**

5.1.2.2. Assemble the relevant knowledge:

In the second step, we will assemble the relevant knowledge which is required for digital circuits. So for digital circuits, we have the following required knowledge:

- Logic circuits are made up of wires and gates.
- Signal flows through wires to the input terminal of the gate, and each gate produces the corresponding output which flows further.

- In this logic circuit, there are four types of gates used: **AND, OR, XOR, and NOT**.
- All these gates have one output terminal and two input terminals (except NOT gate, it has one input terminal).

5.1.2.3. Decide on vocabulary:

The next step of the process is to select functions, predicate, and constants to represent the circuits, terminals, signals, and gates. Firstly we will distinguish the gates from each other and from other objects. Each gate is represented as an object which is named by a constant, such as, **Gate(X1)**. The functionality of each gate is determined by its type, which is taken as constants such as **AND, OR, XOR, or NOT**. Circuits will be identified by a predicate: **Circuit (C1)**.

For the terminal, we will use predicate: **Terminal(x)**.

For gate input, we will use the function **In(1, X1)** for denoting the first input terminal of the gate, and for output terminal we will use **Out (1, X1)**.

The function **Arity(c, i, j)** is used to denote that circuit c has i input, j output.

The connectivity between gates can be represented by predicate **Connect(Out(1, X1), In(1, X1))**.

We use a unary predicate **On (t)**, which is true if the signal at a terminal is on.

5.1.2.4. Encode general knowledge about the domain:

To encode the general knowledge about the logic circuit, we need some following rules:

- If two terminals are connected then they have the same input signal, it can be represented as:

∀ t1, t2 Terminal (t1) ∧ Terminal (t2) ∧ Connect (t1, t2) → Signal (t1) = Signal (2).

- Signal at every terminal will have either value 0 or 1, it will be represented as:

  ∀ t Terminal (t) →Signal (t) = 1 ∨Signal (t) = 0.

- Connect predicates are commutative:

  ∀ t1, t2 Connect(t1, t2) → Connect (t2, t1).

- Representation of types of gates:

  ∀ g Gate(g) ∧ r = Type(g) → r = OR ∨r = AND ∨r = XOR ∨r = NOT.

- Output of AND gate will be zero if and only if any of its input is zero.

  ∀ g Gate(g) ∧ Type(g) = AND →Signal (Out(1, g))= 0 ⇔ ∃n Signal (In(n, g))= 0.

- Output of OR gate is 1 if and only if any of its input is 1:

  ∀ g Gate(g) ∧ Type(g) = OR → Signal (Out(1, g))= 1 ⇔ ∃ n Signal (In(n, g))= 1

- Output of XOR gate is 1 if and only if its inputs are different:

  ∀ g Gate(g) ∧ Type(g) = XOR → Signal (Out(1, g)) = 1 ⇔ Signal (In(1, g)) ≠ Signal (In(2, g)).

- Output of NOT gate is invert of its input:

  ∀ g Gate(g) ∧ Type(g) = NOT → Signal (In(1, g)) ≠ Signal (Out(1, g)).

- All the gates in the above circuit have two inputs and one output (except NOT gate).
- ∀ g Gate(g) ∧ Type(g) = NOT → Arity(g, 1, 1)
- ∀ g Gate(g) ∧ r =Type(g) ∧ (r= AND ∨r= OR ∨r= XOR) → Arity (g, 2, 1).

- All gates are logic circuits:

∀ g Gate(g) → Circuit (g).

5.1.2.5. Encode a description of the problem instance:

Now we encode problem of circuit C1, firstly we categorize the circuit and its gate components. This step is easy if ontology about the problem is already thought. This step involves the writing simple atomics sentences of instances of concepts, which is known as ontology.

For the given circuit C1, we can encode the problem instance in atomic sentences as below:

Since in the circuit there are two XOR, two AND, and one OR gate so atomic sentences for these gates will be:

For XOR gate: Type(x1)= XOR, Type(X2) = XOR

For AND gate: Type(A1) = AND, Type(A2)= AND

For OR gate: Type (O1) = OR.

And then represent the connections between all the gates.

5.1.2.6. Pose queries to the inference procedure and get answers:

In this step, we will find all the possible set of values of all the terminal for the adder circuit. The first query will be:

What should be the combination of input which would generate the first output of circuit C1, as 0 and a second output to be 1?

∃ i1, i2, i3 Signal (In(1, C1))=i1 ∧ Signal (In(2, C1))=i2 ∧ Signal (In(3, C1))= i3
∧ Signal (Out(1, C1)) =0 ∧ Signal (Out(2, C1))=1

5.1.2.7. Debug the knowledge base:

Now we will debug the knowledge base, and this is the last step of the complete process. In this step, we will try to debug the issues of knowledge base.

In the knowledge base, we may have omitted assertions like 1 ≠ 0.

Check your Progress-1

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the chapter.

i. What is knowledge-engineering?

…………………………………………………………………

…………………………………………………….................

…………………………………………………………………

…………………………………………………………………

ii. How to identify the task in Knowledge Engineering Process?

…………………………………………………………………

…………………………………………………………………

…………………………………………………….................

…………………………………………………………………

5.2 Handling Uncertain Process

Uncertainty:

Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

Causes of uncertainty:

Following are some leading causes of uncertainty to occur in the real world.

1. Information occurred from unreliable sources.
2. Experimental Errors
3. Equipment fault
4. Temperature variation
5. Climate change.

Probabilistic reasoning:

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.

In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI:

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- **Bayes' rule**
- **Bayesian Statistics**

As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms:

Probability: Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

$0 \leq P(A) \leq 1$, where P(A) is the probability of an event A.

$P(A) = 0$, indicates total uncertainty in an event A.

$P(A) = 1$, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

$$\textbf{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- $P(\neg A)$ = probability of a not happening event.
- $P(\neg A) + P(A) = 1$.

Event: Each possible outcome of a variable is called an event.

Sample space: The collection of all possible events is called sample space.

Random variables: Random variables are used to represent the events and objects in the real world.

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Conditional probability:

Conditional probability is a probability of occurring an event when another event has already happened.

Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

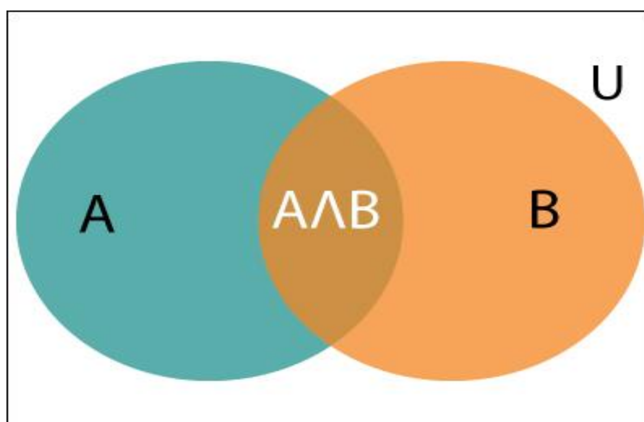
Where P($A \wedge B$)= Joint probability of a and B

P(B)= Marginal probability of B.

If the probability of A is given and we need to find the probability of B, then it will be given as:

$$P(B|A) = \frac{P(A \wedge B)}{P(A)}$$

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of **P(A∧*B*) by P(B)**.



Example:

In a class, there are 70% of the students who like English and 40% of the students who likes English and mathematics, and then what is the percent of students those who like English also like mathematics?

Solution:

Let, A is an event that a student likes Mathematics

B is an event that a student likes English.

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

Hence, 57% are the students who like English also like Mathematics.

Check your Progress-2

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the chapter.

iii. Define Prior Probability.

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

iv. Define Posterior Probability.

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

End Unit Exercise

1. Define Knowledge Engineering.
2. Describe the Process of Knowledge Engineering.
3. Define Uncertainty.
4. Explain Probabilistic Reasoning.

Answers to check your progress:

i. The process of constructing a knowledge-base in first-order logic is called as knowledge- engineering.

ii. The first step of the process is to identify the task, and for the digital circuit, there are various reasoning tasks. At the first level or highest level, we will examine the functionality of the circuit. At the second level, we will examine the circuit structure details.

iii. **Prior probability:** The prior probability of an event is probability computed before observing new information.

iv. **Posterior Probability:** The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Suggested Reading:

1. Stuart Russel, Peter Norvig, "Artificial Intelligence: A Modern Approach – 3 /e", 2014. Pearson Education.
2. Stuart Russel, Peter Norvig, "Artificial Intelligence: A Modern Approach – 2 /e", 2003. Pearson Education.
3. Elaine Rich, Kevin Knight, "Artificial Intelligence" 2/e, 1991, TMH.
4. Dan W. Patterson, "Introduction to Artificial Intelligence & Expert Systems", Seventh Indian Reprint 1999, EEE, PHI.

UNIT VI

6.1 BAYESIAN NETWORK

6.1.1 What are Artificial Neural Networks (ANNs)?

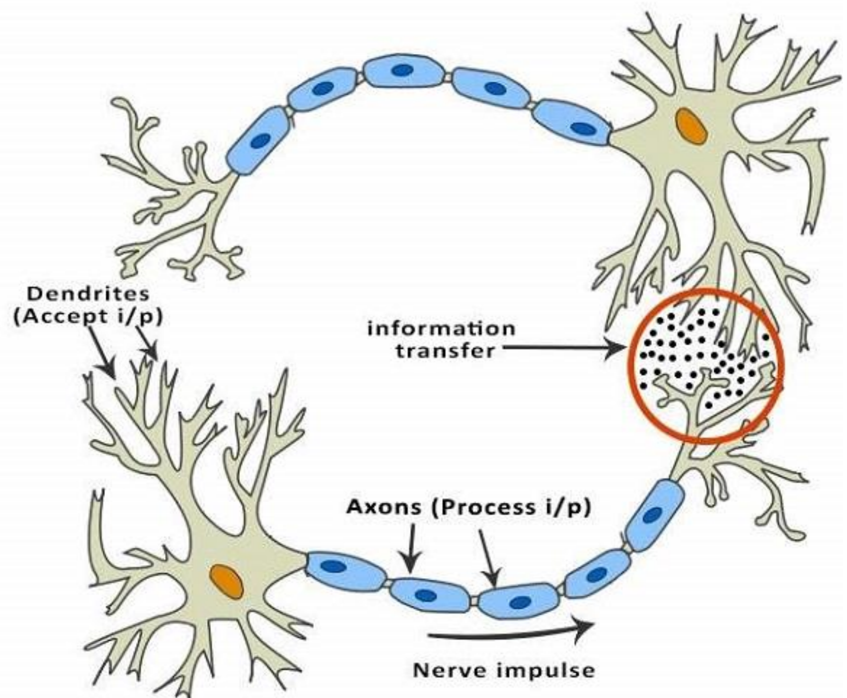
The inventor of the first neuro computer, Dr. Robert Hecht-Nielsen, defines a neural network as −

"A computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.”

Basic Structure of ANNs

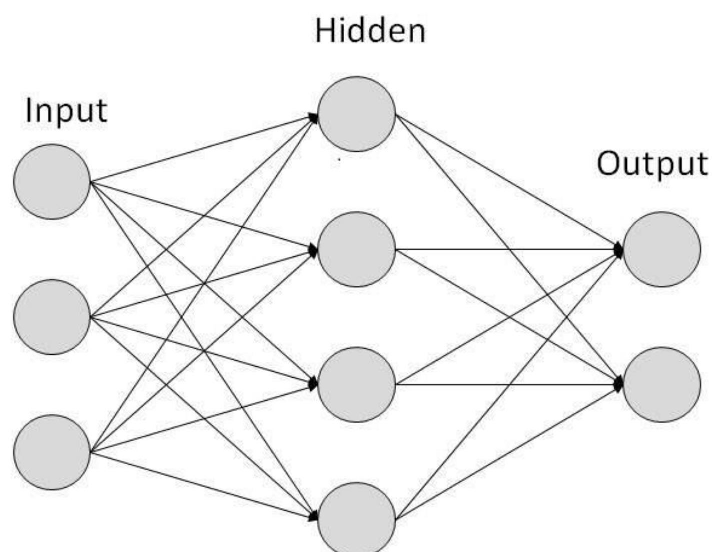
The idea of ANNs is based on the belief that working of human brain by making the right connections can be imitated using silicon and wires as living **neurons** and **dendrites**.

The human brain is composed of 86 billion nerve cells called **neurons.** They are connected to other thousand cells by **Axons.** Stimuli from external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to other neuron to handle the issue or does not send it forward.



ANNs are composed of multiple **nodes**, which imitate biological **neurons** of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its **activation** or **node value.**

Each link is associated with **weight.** ANNs are capable of learning, which takes place by altering weight values. The following illustration shows a simple ANN −

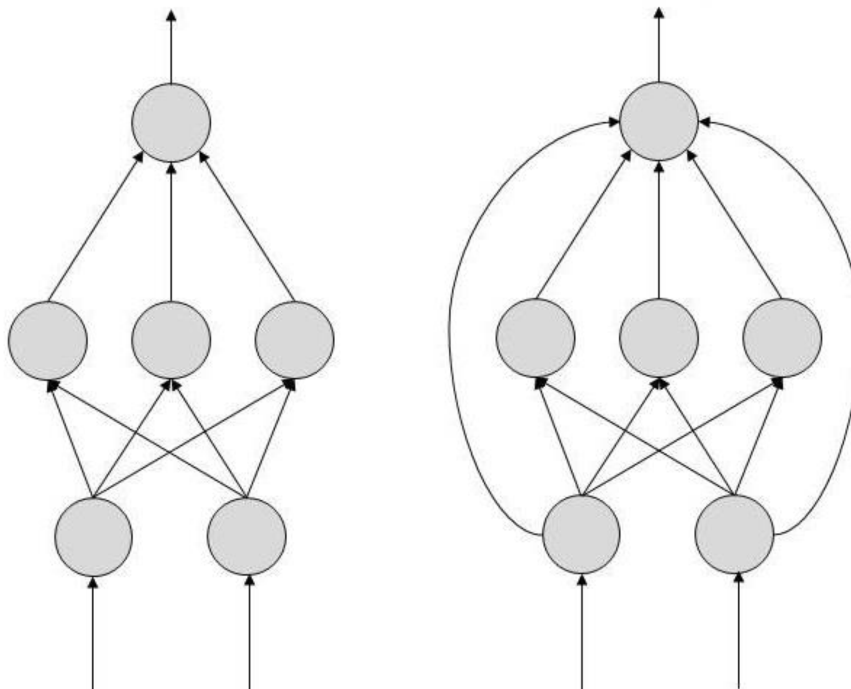


6.1.2. Types of Artificial Neural Networks

There are two Artificial Neural Network topologies − **FeedForward** and **Feedback.**

FeedForward ANN

In this ANN, the information flow is unidirectional. A unit sends information to other unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation/recognition/classification. They have fixed inputs and outputs.

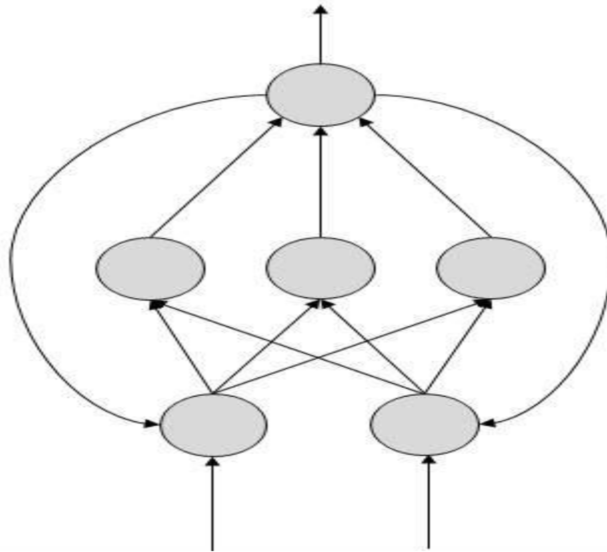


Feedback ANN

Here, feedback loops are allowed. They are used in content addressable memories.

Working of ANNs

In the topology diagrams shown, each arrow represents a connection between two neurons and indicates the pathway for the flow of information. Each connection has a weight, an integer number that controls the signal between the two neurons.



If the network generates a “good or desired” output, there is no need to adjust the weights. However, if the network generates a “poor or undesired” output or an error, then the system alters the weights in order to improve subsequent results.

6.1.3. Machine Learning in ANNs

ANNs are capable of learning and they need to be trained. There are several learning strategies −

- **Supervised Learning** − It involves a teacher that is scholar than the ANN itself. For example, the teacher feeds some example data about which the teacher already knows the answers.

For example, pattern recognizing. The ANN comes up with guesses while recognizing. Then the teacher provides the ANN with the answers. The network then compares it guesses with the teacher’s “correct” answers and makes adjustments according to errors.

- **Unsupervised Learning** − It is required when there is no example data set with known answers. For example, searching for a hidden pattern. In this case, clustering i.e. dividing a set of elements into groups according to some unknown pattern is carried out based on the existing data sets present.
- **Reinforcement Learning** − This strategy built on observation. The ANN makes a decision by observing its environment. If the observation is negative, the network adjusts its weights to be able to make a different required decision the next time.

Back Propagation Algorithm

It is the training or learning algorithm. It learns by example. If you submit to the algorithm the example of what you want the network to do, it changes the network's weights so that it can produce desired output for a particular input on finishing the training.

Back Propagation networks are ideal for simple Pattern Recognition and Mapping Tasks.

6.1.4. Bayes' theorem in Artificial intelligence

Bayes' theorem:

Bayes' theorem is also known as **Bayes' rule, Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.In probability theory, it relates the conditional probability and marginal probabilities of two random events. Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).

Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

Example: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.

Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:

As from product rule we can write:

$$P(A \wedge B)= P(A|B) P(B) \text{ or}$$

Similarly, the probability of event B with known event A:

$$P(A \wedge B)= P(B|A) P(A)$$

Equating right hand side of both the equations, we will get:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \quad \text{....(a)}$$

The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.

It shows the simple relationship between joint and conditional probabilities. Here,

P(A|B) is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.

P(A) is called the **prior probability**, probability of hypothesis before considering the evidence

P(B) is called **marginal probability**, pure probability of an evidence.

In the equation (a), in general, we can write P (B) = P(A)*P(B|Ai), hence the Bayes' rule can be written as:

$$P(A_i|B) = \frac{P(A_i)*P(B|A_i)}{\sum_{i=1}^{k} P(A_i)*P(B|A_i)}$$

Where A_1, A_2, A_3,........, A_n is a set of mutually exclusive and exhaustive events.

Applying Bayes' rule:

Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(*B*), and P(A). This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one. Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause})\,P(\text{cause})}{P(\text{effect})}$$

Example-1:

Question: what is the probability that a patient has diseases meningitis with a stiff neck?

Given Data:

A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:

- The Known probability that a patient has meningitis disease is 1/30,000.
- The Known probability that a patient has a stiff neck is 2%.

Let a be the proposition that patient has stiff neck and b be the proposition that patient has meningitis. , so we can calculate the following as:

P(a|b) = 0.8

P(b) = 1/30000

P(a)= .02

$$\mathbf{P(b|a)} = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8*(\frac{1}{30000})}{0.02} = 0.001333333.$$

Hence, we can assume that 1 patient out of 750 patients has meningitis disease with a stiff neck.

Example-2:

Question: From a standard deck of playing cards, a single card is drawn. The probability that the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.

Solution:

$$\mathbf{P(king|face)} = \frac{P(Face|king) * P(King)}{P(Face)} \quad \text{.......(i)}$$

P(king): probability that the card is King= 4/52= 1/13

P(face): probability that a card is a face card= 3/13

P(Face|King): probability of face card when we assume it is a king = 1

Putting all values in equation (i) we will get:

$$P(king|face) = \frac{1 * (\frac{1}{13})}{(\frac{3}{13})} = \mathbf{1/3}, \text{ it is a probability that a face card is a ki}$$

Application of Bayes' theorem in Artificial intelligence:

Following are some applications of Bayes' theorem:

- It is used to calculate the next step of the robot when the already executed step is given.
- Bayes' theorem is helpful in weather forecasting.
- It can solve the Monty Hall problem.

6.1.5. Bayesian Networks (BN)

- These are the graphical structures used to represent the probabilistic relationship among a set of random variables. Bayesian networks are also called **Belief Networks** or **Bayes Nets.** BNs reason about uncertain domain.
- In these networks, each node represents a random variable with specific propositions. For example, in a medical diagnosis domain, the node Cancer represents the proposition that a patient has cancer.

The edges connecting the nodes represent probabilistic dependencies among those random variables. If out of two nodes, one is affecting the other then they must be directly connected in the directions of the effect.

The strength of the relationship between variables is quantified by the probability associated with each node.

There is an only constraint on the arcs in a BN that you cannot return to a node simply by following directed arcs. Hence the BNs are called Directed Acyclic Graphs (DAGs).

BNs are capable of handling multivalued variables simultaneously. The BN variables are composed of two dimensions −

- Range of prepositions
- Probability assigned to each of the prepositions.

Consider a finite set $X = \{X_1, X_2, \ldots, X_n\}$ of discrete random variables, where each variable X_i may take values from a finite set, denoted by $Val(X_i)$. If there is a directed link from variable X_i to variable, X_j, then variable X_i will be a parent of variable X_j showing direct dependencies between the variables.

The structure of BN is ideal for combining prior knowledge and observed data. BN can be used to learn the causal relationships and understand various problem domains and to predict future events, even in case of missing data.

Building a Bayesian Network

A knowledge engineer can build a Bayesian network. There are a number of steps the knowledge engineer needs to take while building it.

Example problem − *Lung cancer*. A patient has been suffering from breathlessness. He visits the doctor, suspecting he has lung cancer. The doctor knows that barring lung cancer, there are various other possible diseases the patient might have such as tuberculosis and bronchitis.

Gather Relevant Information of Problem

- Is the patient a smoker? If yes, then high chances of cancer and bronchitis.
- Is the patient exposed to air pollution? If yes, what sort of air pollution?
- Take an X-Ray positive X-ray would indicate either TB or lung cancer.

Identify Interesting Variables

The knowledge engineer tries to answer the questions −

- Which nodes to represent?
- What values can they take? In which state can they be?

For now let us consider nodes, with only discrete values. The variable must take on exactly one of these values at a time.

Common types of discrete nodes are −

- **Boolean nodes** − They represent propositions, taking binary values TRUE (T) and FALSE (F).
- **Ordered values** − A node *Pollution* might represent and take values from {low, medium, high} describing degree of a patient's exposure to pollution.
- **Integral values** − A node called *Age* might represent patient's age with possible values from 1 to 120. Even at this early stage, modeling choices are being made.

Possible nodes and values for the lung cancer example −

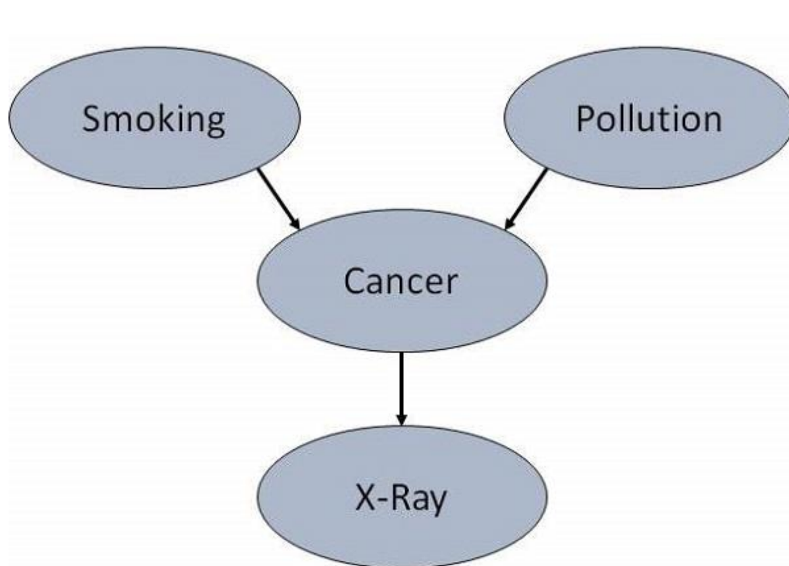
Node Name	Type	Value	Nodes Creation
Polution	Binary	{LOW, HIGH, MEDIUM}	Smoking Pollution Cancer X-Ray
Smoker	Boolean	{TRUE, FASLE}	
Lung-Cancer	Boolean	{TRUE, FASLE}	
X-Ray	Binary	{Positive, Negative}	

Create Arcs between Nodes

Topology of the network should capture qualitative relationships between variables.

For example, what causes a patient to have lung cancer? - Pollution and smoking. Then add arcs from node *Pollution* and node *Smoker* to node *Lung-Cancer*.

Similarly if patient has lung cancer, then X-ray result will be positive. Then add arcs from node *Lung-Cancer* to node *X-Ray*.



Specify Topology

Conventionally, BNs are laid out so that the arcs point from top to bottom. The set of parent nodes of a node X is given by Parents(X).

The *Lung-Cancer* node has two parents (reasons or causes): *Pollution* and *Smoker*, while node *Smoker* is an **ancestor** of node *X-Ray*. Similarly, *X-Ray* is a child (consequence or effects) of node *Lung-Cancer* and **successor** of nodes *Smoker* and *Pollution.*

Conditional Probabilities

Now quantify the relationships between connected nodes: this is done by specifying a conditional probability distribution for each node. As only discrete variables are considered here, this takes the form of a **Conditional Probability Table (CPT).**

First, for each node we need to look at all the possible combinations of values of those parent nodes. Each such combination is called an **instantiation** of the parent set. For each distinct instantiation of parent node values, we need to specify the probability that the child will take.

For example, the *Lung-Cancer* node's parents are *Pollution* and *Smoking.* They take the possible values = { (H,T), (H,F), (L,T), (L,F)}. The CPT specifies the probability of cancer for each of these cases as <0.05, 0.02, 0.03, 0.001> respectively.

Each node will have conditional probability associated as follows −

Smoking
P(S = T)
0.30

Pollution
P(P = L)
0.90

Lung-Cancer		
P	**S**	**P (C = T|P, S)**
H	T	0.05
H	F	0.02
L	T	0.03
L	F	0.001

X-Ray	
C	**X = (Pos | C)**
T	0.90
F	0.20

6.1.6. Bayesian Belief Network in artificial intelligence

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.

Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network.

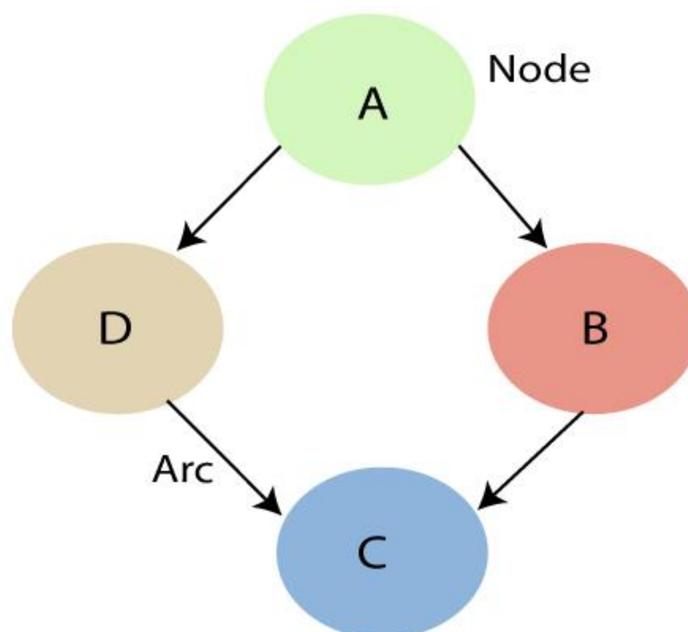
It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- **Directed Acyclic Graph**
- **Table of conditional probabilities.**

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:



- Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.
- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph.

These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other

- **In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.**
- **If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.**
- **Node C is independent of node A.**

The Bayesian network has mainly two components:

- **Causal Component**
- **Actual numbers**

Each node in the Bayesian network has condition probability distribution $\mathbf{P(X_i \mid Parent(X_i))}$, which determines the effect of the parent on that node.

Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables x1, x2, x3,....., xn, then the probabilities of a different combination of x1, x2, x3.. xn, are known as Joint probability distribution.

$\mathbf{P[x_1, x_2, x_3,....., x_n]}$, it can be written as the following way in terms of the joint probability distribution.

$\mathbf{= P[x_1 \mid x_2, x_3,....., x_n]P[x_2, x_3,....., x_n]}$

$\mathbf{= P[x_1 \mid x_2, x_3,....., x_n]P[x_2 \mid x_3,....., x_n]....P[x_{n-1} \mid x_n]P[x_n].}$

In general for each variable Xi, we can write the equation as:

$$P(X_i \mid X_{i-1},........., X_1) = P(X_i \mid Parents(X_i))$$

Let's understand the Bayesian network through an example by creating a directed acyclic graph:

Example:

Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

Problem:

Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

Solution:

- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.
- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table or CPT.
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- In CPT, a boolean variable with k boolean parents contains 2^K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

List of all events occurring in this network:

- **Burglary (B)**
- **Earthquake(E)**
- **Alarm(A)**
- **David Calls(D)**
- **Sophia calls(S)**

We can write the events of problem statement in the form of probability: **P[D, S, A, B, E]**, can rewrite the above probability statement using joint probability distribution:

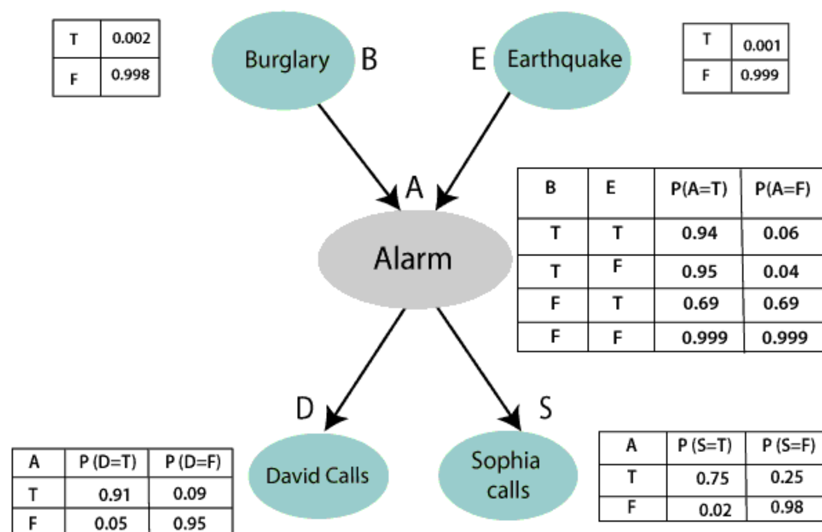
P[D, S, A, B, E]= P[D | S, A, B, E]. P[S, A, B, E]

=P[D | S, A, B, E]. P[S | A, B, E]. P[A, B, E]

= P [D| A]. P [S| A, B, E]. P[A, B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B |E]. P[E]



Let's take the observed probability for the Burglary and earthquake component:

P(B= True) = 0.002, which is the probability of burglary.

P(B= False)= 0.998, which is the probability of no burglary.

P(E= True)= 0.001, which is the probability of a minor earthquake

P(E= False)= 0.999, Which is the probability that an earthquake not occurred.

We can provide the conditional probabilities as per the below tables:

Conditional probability table for Alarm A:

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A= True)	P(A= False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

P(S, D, A, ¬B, ¬E) = P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E).

= 0.75* 0.91* 0.001* 0.998*0.999

= 0.00068045.

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

The semantics of Bayesian Network:

There are two ways to understand the semantics of the Bayesian network, which is given below:

1. To understand the network as the representation of the Joint probability distribution.

It is helpful to understand how to construct the network.

2. To understand the network as an encoding of a collection of conditional independence statements.

It is helpful in designing inference procedure.

6.1.6. Applications of Neural Networks

They can perform tasks that are easy for a human but difficult for a machine −

- **Aerospace** − Autopilot aircrafts, aircraft fault detection.
- **Automotive** − Automobile guidance systems.
- **Military** − Weapon orientation and steering, target tracking, object discrimination, facial recognition, signal/image identification.

- **Electronics** − Code sequence prediction, IC chip layout, chip failure analysis, machine vision, voice synthesis.
- **Financial** − Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, portfolio trading program, corporate financial analysis, currency value prediction, document readers, credit application evaluators.
- **Industrial** − Manufacturing process control, product design and analysis, quality inspection systems, welding quality analysis, paper quality prediction, chemical product design analysis, dynamic modeling of chemical process systems, machine maintenance analysis, project bidding, planning, and management.
- **Medical** − Cancer cell analysis, EEG and ECG analysis, prosthetic design, transplant time optimizer.
- **Speech** − Speech recognition, speech classification, text to speech conversion.
- **Telecommunications** − Image and data compression, automated information services, real-time spoken language translation.
- **Transportation** − Truck Brake system diagnosis, vehicle scheduling, routing systems.
- **Software** − Pattern Recognition in facial recognition, optical character recognition, etc.
- **Time Series Prediction** − ANNs are used to make predictions on stocks and natural calamities.
- **Signal Processing** − Neural networks can be trained to process an audio signal and filter it appropriately in the hearing aids.
- **Control** − ANNs are often used to make steering decisions of physical vehicles.
- **Anomaly Detection** − As ANNs are expert at recognizing patterns, they can also be trained to generate an output when something unusual occurs that misfits the pattern.

Check your Progress-1

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the chapter.

i. Define ANN.

…………………………………………………………………

…………………………………………………………....................

…………………………………………………………………

…………………………………………………………………

ii. Write the two Components of Bayesian Network.

…………………………………………………………………

…………………………………………………………………

…………………………………………………………....................

…………………………………………………………………

iii. Write about the semantics of Bayesian Network .

…………………………………………………………………

…………………………………………………………………

…………………………………………………………....................

…………………………………………………………………

6.2 LEARNING

6.2.1 What is learning?

According to **Herbert Simon**, learning denotes changes in a system that enable a system to do the same task more efficiently the next time.

Arthur Samuel stated that, "Machine learning is the subfield of computer science that gives computers the ability to learn without being explicitly programmed ".

In 1997, **Mitchell** proposed that, "A computer program is said to learn from experience **'E'** with respect to some class of tasks **'T'** and performance measure **'P'**, if its performance at tasks in **'T'**, as measured by **'P'**, improves with experience **E** ".

The main purpose of machine learning is to study and design the algorithms that can be used to produce the predicates from the given dataset.

Besides these, the machine learning includes the agents' percepts for acting as well as to improve their future performance.

The following tasks must be learned by an agent.

- To predict or decide the result state for an action.
- To know the values for each state (understand which state has high or low vale).
- To keep record of relevant percepts.

Learning –

- It is the activity of gaining knowledge or skill by studying, practising, being taught, or experiencing something.
- Learning enhances the awareness of the subjects of the study.

The ability of learning is possessed by humans, some animals, and AI-enabled systems.

Learning is categorized as −

- **Auditory Learning** − It is learning by listening and hearing. For example, students listening to recorded audio lectures.
- **Episodic Learning** − To learn by remembering sequences of events that one has witnessed or experienced. This is linear and orderly.
- **Motor Learning** − It is learning by precise movement of muscles. For example, picking objects, Writing, etc.
- **Observational Learning** − To learn by watching and imitating others. For example, child tries to learn by mimicking her parent.
- **Perceptual Learning** − It is learning to recognize stimuli that one has seen before. For example, identifying and classifying objects and situations.
- **Relational Learning** − It involves learning to differentiate among various stimuli on the basis of relational properties, rather than absolute properties. For Example, Adding 'little less' salt at the time of cooking potatoes that came up salty last time, when cooked with adding say a tablespoon of salt.
- **Spatial Learning** − It is learning through visual stimuli such as images, colors, maps, etc. For Example, A person can create roadmap in mind before actually following the road.
- **Stimulus-Response Learning** − It is learning to perform a particular behavior when a certain stimulus is present. For example, a dog raises its ear on hearing doorbell.

6.2.2 Various forms of learning

6.2.2.1. Rote learning

Rote learning is possible on the basis of memorization.

This technique mainly focuses on memorization by avoiding the inner complexities. So, it becomes possible for the learner to recall the stored knowledge. **For example:** When a learner learns a poem or song by reciting or repeating it, without knowing the actual meaning of the poem or song.

6.2.2.2. Induction learning (Learning by example).

- Induction learning is carried out on the basis of supervised learning.
- In this learning process, a general rule is induced by the system from a set of observed instance.
- However, class definitions can be constructed with the help of a classification method.

For Example:

Consider that **'*f*'** is the target function and example is a pair (x *f*(x)), where 'x' is input and *f*(x) is the output function applied to 'x'.

Given problem: Find hypothesis h such as h ≈ *f*

So, in the following fig-a, points (x,y) are given in plane so that y = *f*(x), and the task is to find a function h(x) that fits the point well.

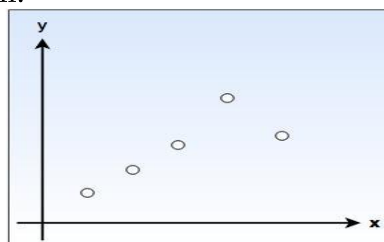


Fig- a

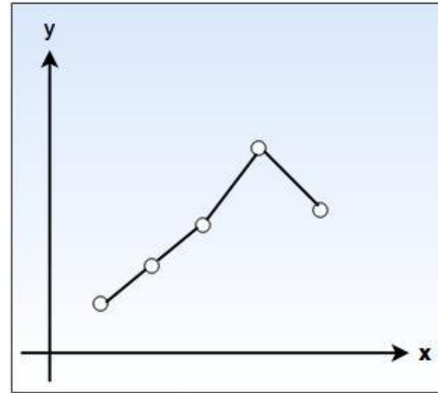


Fig-b

In fig-b, a piecewise-linear 'h' function is given, while the fig-c shows more complicated 'h' function.

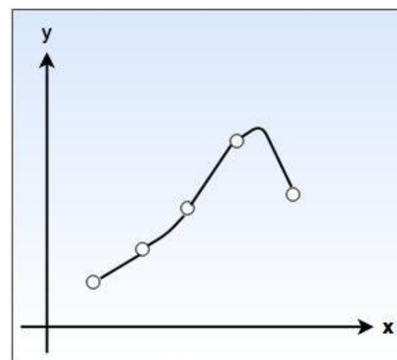


Fig-c

Both the functions agree with the example points, but differ with the values of 'y' assigned to other x inputs.

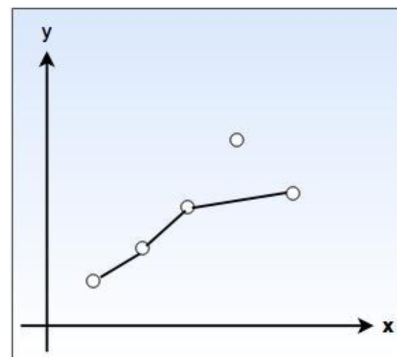


Fig-d

As shown in fig.(d), we have a function that apparently ignores one of the example points, but fits others with a simple function. The true/ is unknown, so there are many choices for h, but without further knowledge, we have no way to prefer (b), (c), or (d).

6.2.2.3. Learning by taking advice

- This type is the easiest and simple way of learning.
- In this type of learning, a programmer writes a program to give some instructions to perform a task to the computer. Once it is learned (i.e. programmed), the system will be able to do new things.
- Also, there can be several sources for taking advice such as humans (experts), internet etc.
- However, this type of learning has a more necessity of inference than rote learning.
- As the stored knowledge in knowledge base gets transformed into an operational form, the reliability of the knowledge source is always taken into consideration.

6.2.2.4. Explanation-based learning (EBL)

Explanation-based learning (EBL) deals with an idea of single-example learning.

This type of learning usually requires a substantial number of training instances but there are two difficulties in this:

I. it is difficult to have such a number of training instances

ii. Sometimes, it may help us to learn certain things effectively, specially when we have enough knowledge.

Hence, it is clear that instance-based learning is more data-intensive, data-driven while EBL is more knowledge-intensive,

On the basis of the given goal concept, an operationality criteria and domain theory, it "generalizes" the training example to describe the goal concept and to satisfy the operationality criteria (which are usually a set of rules that describe relationships between objects and actions in a domain).

Thus, several applications are possible for the knowledge acquisition and engineering aspects.

Learning in Problem Solving:

- Humans have a tendency to learn by solving various real world problems.
- The forms or representation, or the exact entity, problem solving principle is based on reinforcement learning.
- Therefore, repeating certain action results in desirable outcome while the action is avoided if it results into undesirable outcomes.
- As the outcomes have to be evaluated, this type of learning also involves the definition of a utility function. This function shows how much is a particular outcome worth?
- There are several research issues which include the identification of the learning rate, time and algorithm complexity, convergence, representation (frame and qualification problems), handling of uncertainty (ramification problem), adaptivity and "unlearning" etc.
- In reinforcement learning, the system (and thus the developer) know the desirable outcomes but does not know which actions result into desirable outcomes.

In such a problem or domain, the effects of performing the actions are usually compounded with side-effects.

- Thus, it becomes impossible to specify the actions to be performed in accordance to the given parameters.
- Q-Learning is the most widely used reinforcement learning algorithm.

The main part of an algorithm is a simple value iteration update. For each state 'S', from the state set S, and for each action, a, from the action set 'A', it is possible to calculate an update to its expected reduction reward value, with the following expression:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t (s_t, a_t) [r_t + \gamma \max_a Q (s_{t+1}, a) - Q(s_t, a_t)]$$

where r_t is a real reward at time t, $\alpha_t(s,a)$ are the learning rates such that $0 \leq \alpha_t(s,a) \leq 1$, and γ is the discount factor such that $0 \leq \gamma < 1$.

6.2.3 Machine Learning in Artificial Intelligence

At its core, **machine learning is simply a way of achieving AI**. Machine learning is an application of artificial intelligence (AI) that enables systems to learn and advance based on experience without being clearly programmed. Machine learning focuses on the development of computer programs that can access data and use it for their own learning.

There are 4 types of machine learning

1. Supervised learning
2. Unsupervised learning
3. Semi-supervised learning
4. Reinforced learning

6.2.3.1 Supervised learning

Supervised machine learning can take what it has learned in the past and apply that to new data using labeled examples to predict future patterns and events. It learns by explicit example.

Supervised learning requires that the algorithm's possible outputs are already known and that the data used to train the algorithm is already labeled with correct answers. It's like teaching a child that 2+2=4 or showing an image of a dog and teaching the child that it is called a dog. The approach to supervised machine learning is essentially the same – it is presented with all the information it needs to reach pre-determined conclusions. It learns how to reach the conclusion, just like a child would learn how to reach the total of '5' and the few, pre-determined ways to get there, for example, 2+3 and 1+4. If you were to present 6+3 as a way to get to 5, that would be determined as incorrect. Errors would be found and adjusted.



The algorithm learns by comparing its actual output with correct outputs to find errors. It then modifies the model accordingly.

Supervised learning is commonly used in applications where **historical data predicts likely future events.** Using the previous example, if 6+3 is the most common erroneous way to get to 5, the machine can predict that when someone inputs 6+3, after the correct answer of 9, 5 would be the second most commonly expected result. We can also consider an everyday example – it can foresee when credit card transactions are likely to be fraudulent or which insurance customer is more likely to put forward a claim.

Supervised learning is further divided into:

1. Classification
2. Regression

6.2.3.2. Unsupervised learning

Supervised learning tasks find patterns where we have a dataset of “right answers” to learn from. **Unsupervised learning tasks find patterns where we don’t.** This may be because the “right answers” are unobservable, or infeasible to obtain, or maybe for a given problem, there isn’t even a “right answer” per se.

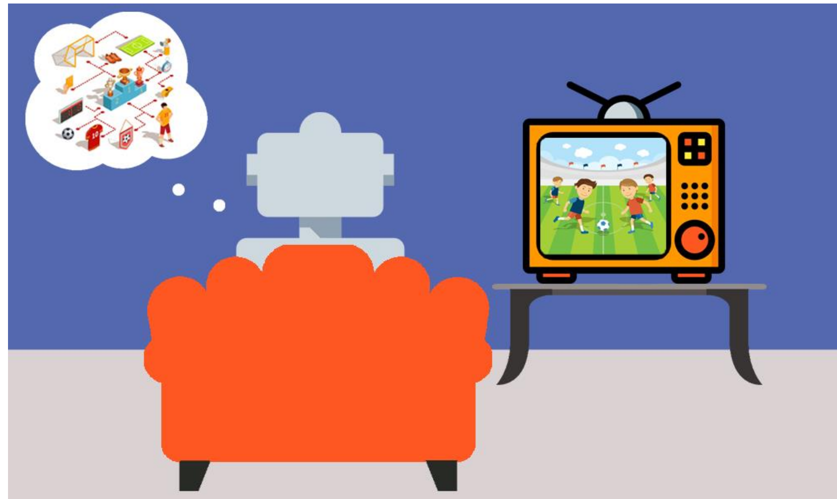
Unsupervised learning is used against data without any historical labels. The system is not given a pre-determined set of outputs or correlations between inputs and outputs or a “correct answer.” The algorithm must figure out what it is seeing by itself, it has no storage of reference points. The goal is to explore the data and find some sort of patterns of structure.

Unsupervised learning works well when the data is transactional. For example, identifying pockets of customers with similar characteristics who can then be targeted in marketing campaigns.

Unsupervised machine learning is a more complex process and has been used far fewer times than supervised machine learning. But it’s exactly for this reason that there is so much buzz around the future of AI. Advances in unsupervised ML are seen as the future of AI because it moves away from narrow AI and closer to AGI (‘artificial general intelligence’ that we discussed a few paragraphs earlier). If you’ve ever heard someone talking about computers teaching themselves, this is essentially what they are referring to.

In unsupervised learning, neither a training data set nor a list of outcomes is provided. The AI enters the problem blind – with only its faultless logical operations to guide it.

Imagine yourself as a person that has never heard of or seen any sport being played. You get taken to a football game and left to figure out what it is that you are observing. You can’t refer to your knowledge of other sports and try to draw up similarities and differences that will eventually boil down to an understanding of football. You have nothing but your cognitive ability. Unsupervised learning places the AI in an equivalent of this situation and leaves it to learn using only it’s on/off logic mechanisms that are used in all computer systems.



6.2.3 3. Semi-supervised learning (SSL)

Semi-supervised learning falls somewhere in the middle of supervised and unsupervised learning. It is used because many problems that AI is used to solving require a balance of both approaches.

In many cases the reference data needed for solving the problem is available, but it is either incomplete or somehow inaccurate. This is when semi-supervised learning is summoned for help since it is able to access the available reference data and then use unsupervised learning techniques to do its best to fill the gaps.

Unlike supervised learning which uses labeled data and unsupervised which is given no labeled data at all, SSL uses both. More often than not the scales tip in favor of unlabelled data since it is cheaper and easier to acquire, leaving the volume of available labeled data in the minority. The AI learns from the labeled data to then make a judgment on the unlabelled data and find patterns, relationships and structures.

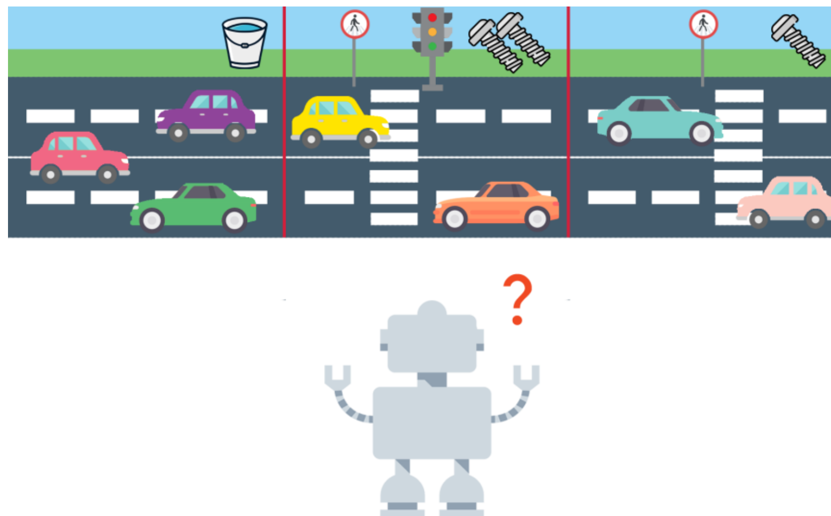
SSL is also useful in reducing human bias in the process. A fully labeled, supervised learning AI has been labeled by a human and thus poses the risk of results potentially being skewed due to improper labeling. With SSL, including a lot of unlabelled data in the training process often improves the precision of the end result while time and cost are reduced. It enables data scientists to access and use lots of unlabelled data without having to face the insurmountable task of assigning information and labels to each one.

6.2.3.4. Reinforcement learning

Reinforcement learning is a type of dynamic programming that trains algorithms using **a system of reward and punishment.**

A reinforcement learning algorithm, or agent, learns by interacting with its environment. It receives rewards by performing correctly and penalties for doing so incorrectly. Therefore, it learns without having to be directly taught by a human – **it learns by seeking the greatest reward and minimizing penalty.** This learning is tied to a context because what may lead to maximum reward in one situation may be directly associated with a penalty in another.

This type of learning consists of three components: the agent (the AI learner/decision maker), the environment (everything the agent has interaction with) and actions (what the agent can do). The agent will reach the goal much faster by finding the best way to do it – and that is the goal – maximizing the reward, minimizing the penalty and figuring out the best way to do so.



Machines and software agents learn to determine the perfect behavior within a specific context, to maximize its performance and reward. Learning occurs via reward feedback which is known as the reinforcement signal. An everyday example of training pets to relieve themselves outside is a simple way to illustrate this. The goal is getting the pet into the habit of going outside rather than in the house. The training then involves rewards and punishments intended for the pet’s learning. It gets a treat for going outside or has its nose rubbed in its mess if it fails to do so.

Reinforcement learning tends to be used for gaming, robotics and navigation. The algorithm discovers which steps lead to the maximum rewards through a process of trial and error. When this is repeated, the problem is known as a **Markov Decision Process.**

Facebook's News Feed is an example most of us will be able to understand. Facebook uses machine learning to personalize people's feeds. If you frequently read or "like" a particular friend's activity, the News Feed will begin to bring up more of that friend's activity more often and nearer to the top. Should you stop interacting with this friend's activity in the same way, the data set will be updated and the News Feed will consequently adjust.

6.2.4 Deep Learning

Deep learning is a **specialized form of machine learning**. Deep Learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. It is also known as Deep Neural Learning or Deep Neural Network. Deep learning uses a hierarchical level of artificial neural networks for the machine learning process. These networks are built to resemble the way the human brain functions, with neuron nodes interconnected like a web. While traditional programs build linear networks, the hierarchical function of deep learning systems enables processing data in a nonlinear way.

A standard machine learning workflow starts with manually extracting selected features from images. These features are then used to create a model for categorizing the selected objects. A deep learning workflow differs from this as relevant features are extracted automatically. In addition, deep learning performs "end-to-end learning" – it is given raw data and a task to perform, such as classification, and it learns how to do this by itself.

In machine learning, you manually choose features and a classifier to sort images. With deep learning, feature extraction and modeling steps are automatic.

However, because deep learning is still going through growing pains, there have been a number of concerns raised alongside the vast potential it holds, particularly around the ambition of achieving AGI through deep learning.

- DL is limited when it comes to open-ended reasoning based on real-world common sense and knowledge, meaning that machines wouldn't be able to distinguish between "Tom promised Mary to stop" and "Tom promised to stop Mary".
- Deep learning is self-sufficient and is made up of correlations, rather than abstractions. Problems that deal largely with common sense reasoning are mostly outside of what deep learning can cope with.
- Another problem often linked to deep learning is acquiring biases. If the training data set contains biases, the model will learn and consequently replicate those biases in its conclusions and predictions.

Check your Progress-2

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the chapter.

iv. Define Learning.

…………………………………………………………………

……………………………………………………..................

…………………………………………………………………

…………………………………………………………………

v. Define EBL.

…………………………………………………………………

…………………………………………………………………

……………………………………………………..................

…………………………………………………………………

6.3 PATTERN RECONGINITION

6.3 .1 Introduction

Pattern is everything around in this digital world. A pattern can either be seen physically or it can be observed mathematically by applying algorithms.

Example: The colors on the clothes, speech pattern etc. In computer science, a pattern is represented using vector features values.

What is Pattern Recognition?
Pattern recognition is the process of recognizing patterns by using machine learning algorithm. Pattern recognition can be defined as the classification of data based on knowledge already gained or on statistical information extracted from patterns and/or their representation. One of the important aspects of the pattern recognition is its application potential.

Examples: Speech recognition, speaker identification, multimedia document recognition (MDR), automatic medical diagnosis. In a typical pattern recognition application, the raw data is processed and converted into a form that is amenable for a machine to use. Pattern recognition involves classification and cluster of patterns.

- In classification, an appropriate class label is assigned to a pattern based on an abstraction that is generated using a set of training patterns or domain knowledge. Classification is used in supervised learning.
- Clustering generated a partition of the data which helps decision making, the specific decision making activity of interest to us. Clustering is used in an unsupervised learning.
-

Features may be represented as continuous, discrete or discrete binary variables. A feature is a function of one or more measurements, computed so that it quantifies some significant characteristics of the object.

Example: consider our face then eyes, ears, nose etc are features of the face.
A set of features that are taken together, forms the features vector. Example: In the above example of face, if all the features (eyes, ears, nose etc) taken together then the sequence is feature vector([eyes, ears, nose]). Feature vector is the sequence of a features represented as a d-dimensional column vector.

In case of speech, MFCC (Melfrequency Cepstral Coefficent) is the spectral features of the speech. Sequence of first 13 features forms a feature vector.

Pattern recognition possesses the following features:

- Pattern recognition system should recognize familiar pattern quickly and accurate
- Recognize and classify unfamiliar objects
- Accurately recognize shapes and objects from different angles
- Identify patterns and objects even when partly hidden
- Recognize patterns quickly with ease, and with automaticity.

6.3 .2 Training and Learning in Pattern Recognition

Learning is a phenomena through which a system gets trained and becomes adaptable to give result in an accurate manner. Learning is the most important phase as how well the system performs on the data provided to the system depends on which algorithms used on the data. Entire dataset is divided into two categories, one which is used in training the model i.e. Training set and the other that is used in testing the model after training, i.e. testing set.

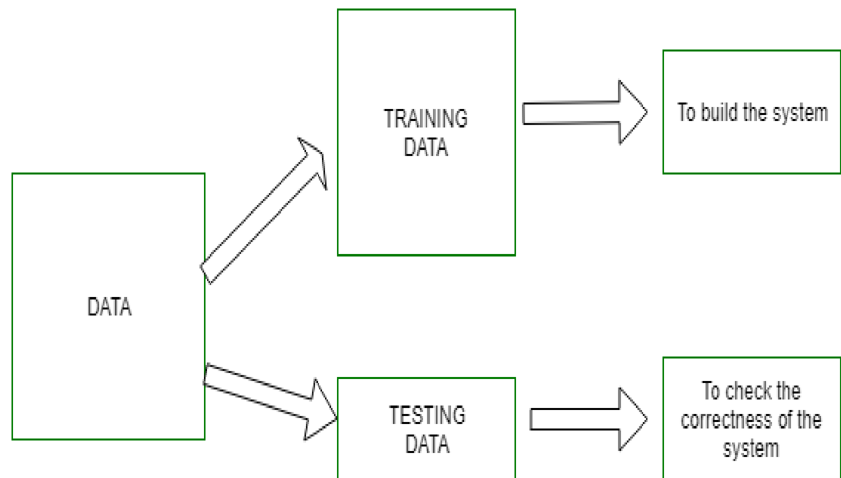
Training set:

- Training set is used to build a model. It consists of the set of images which are used to train the system. Training rules and algorithms used give relevant information on how to associate input data with output decision.
- The system is trained by applying these algorithms on the dataset, all the relevant information is extracted from the data and results are obtained. Generally, 80% of the data of the dataset is taken for training data.

Testing set:

Testing data is used to test the system. It is the set of data which is used to verify whether the system is producing the correct output after being trained or not.

- Generally, 20% of the data of the dataset is used for testing. Testing data is used to measure the accuracy of the system. Example: a system which identifies which category a particular flower belongs to, is able to identify seven category of flowers correctly out of ten and rest others wrong, then the accuracy is 70 %



6.3 .3 Real-time Examples and Explanations:

A pattern is a physical object or an abstract notion. While talking about the classes of animals, a description of an animal would be a pattern. While talking about various types of balls, then a description of a ball is a pattern. In the case balls considered as pattern, the classes could be football, cricket ball, table tennis ball etc. Given a new pattern, the class of the pattern is to be determined. The choice of attributes and representation of patterns is a very important step in pattern classification. A good representation is one which makes use of discriminating attributes and also reduces the computational burden in pattern classification.

An obvious representation of a pattern will be a **vector**. Each element of the vector can represent one attribute of the pattern. The first element of the vector will contain the value of the first attribute for the pattern being considered.

Example:

While representing spherical objects, (25, 1) may be represented as an spherical object with 25 units of weight and 1 unit diameter. The class label can form a part of the vector. If spherical objects belong to class 1, the vector would be (25, 1, 1), where the first element represents the weight of the object, the second element, the diameter of the object and the third element represents the class of the object.

Advantages:

- Pattern recognition solves classification problems
- Pattern recognition solves the problem of fake bio metric detection.
- It is useful for cloth pattern recognition for visually impaired blind people.
- It helps in speaker diarization.
- We can recognize particular object from different angle.

Disadvantages:

- Syntactic Pattern recognition approach is complex to implement and it is very slow process.
- Sometime to get better accuracy, larger dataset is required.
- It cannot explain why a particular object is recognized.
  Example: my face vs my friend's face.

6.3 .4 Applications:

- **Image processing, segmentation and analysis**
  Pattern recognition is used to give human recognition intelligence to machine which is required in image processing.
- **Computer vision**
  Pattern recognition is used to extract meaningful features from given image/video samples and is used in computer vision for various applications like biological and biomedical imaging.
- **Seismic analysis**
  Pattern recognition approach is used for the discovery, imaging and interpretation of temporal patterns in seismic array recordings. Statistical pattern recognition is implemented and used in different types of seismic analysis models.
- **Radar signal classification/analysis**
  Pattern recognition and Signal processing methods are used in various applications of radar signal classifications like AP mine detection and identification.
- **Speech recognition**
  the greatest success in speech recognition has been obtained using pattern recognition paradigms. It is used in various algorithms of speech recognition which tries to avoid the problems of using a phoneme level of description and treats larger units such as words as pattern

- **Finger print identification**

  the fingerprint recognition technique is a dominant technology in the biometric market. A number of recognition methods have been used to perform fingerprint matching out of which pattern recognition approaches is widely used.

6.4 Pattern Recognition: Basics and Design Principles

6.4 .1 Pattern Recognition System

Pattern is everything around in this digital world. A pattern can either be seen physically or it can be observed mathematically by applying algorithms.

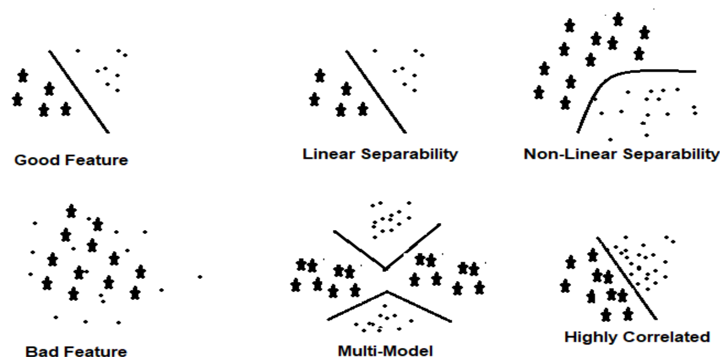
In **Pattern Recognition**, pattern is comprises of the following two fundamental things:

- Collection of observations
- The concept behind the observation

Feature Vector:

The collection of observations is also known as a feature vector. A feature is a distinctive characteristic of a good or service that sets it apart from similar items. **Feature vector** is the combination of n features in n-dimensional column vector. The different classes may have different features values but the same class always has the same features values.

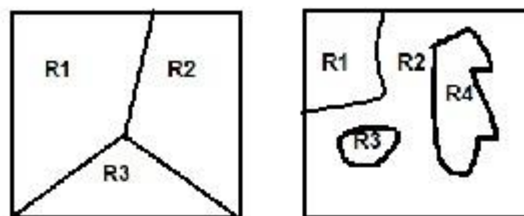
Example:



- Differentiate between good and bad features.
- Feature properties.

Classifier and Decision Boundaries:

1. In a statistical-classification problem, a **decision boundary** is a hyper surface that partitions the underlying vector space into two sets. A decision boundary is the region of a problem space in which the output label of a classifier is ambiguous. **Classifier** is a hypothesis or discrete-valued function that is used to assign (categorical) class labels to particular data points.
2. **Classifier** is used to partition the feature space into class-labeled decision regions. While **Decision Boundaries** are the borders between decision regions.



Classifier and decision boundaries

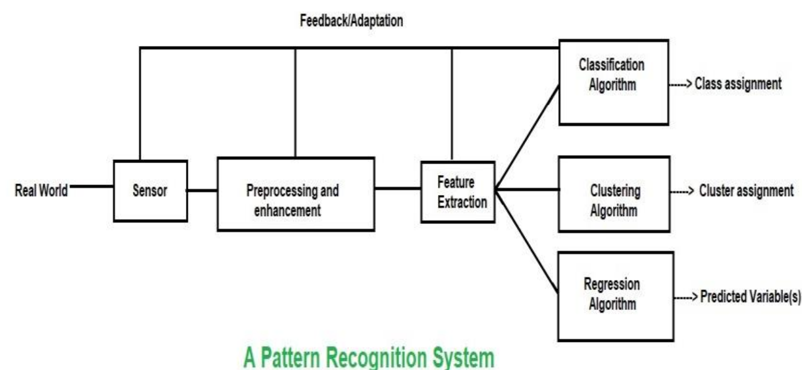
6.4 .2 Components in Pattern Recognition System:

A pattern recognition systems can be partitioned into components. There are five typical components for various pattern recognition systems. These are as following:

- **A Sensor:** A sensor is a device used to measure a property, such as pressure, position, temperature, or acceleration, and respond with feedback.
- **A Preprocessing Mechanism:** Segmentation is used and it is the process of partitioning a data into multiple segments. It can also be defined as the technique of dividing or partitioning a data into parts called segments.
- **A Feature Extraction Mechanism:** feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. It can be manual or automated.

- **A Description Algorithm :** Pattern recognition algorithms generally aim to provide a reasonable answer for all possible inputs and to perform "most likely" matching of the inputs, taking into account their statistical variation

- **A Training Set:** Training data is a certain percentage of an overall dataset along with testing set. As a rule, the better the training data, the better the algorithm or classifier performs.



A Pattern Recognition System

6.4 .3 Design Principles of Pattern Recognition

In pattern recognition system, for recognizing the pattern or structure two basic approaches are used which can be implemented in different techniques. These are –

- Statistical Approach and
- Structural Approach

Statistical Approach:

Statistical methods are mathematical formulas, models, and techniques that are used in the statistical analysis of raw research data. The application of statistical methods extracts information from research data and provides different ways to assess the robustness of research outputs.

Two main statistical methods are used:

1. **Descriptive Statistics:** It summarizes data from a sample using indexes such as the mean or standard deviation.
2. **Inferential Statistics:** It draws conclusions from data that are subject to random variation.

Structural Approach:

The Structural Approach is a technique wherein the learner masters the pattern of sentence. Structures are the different arrangements of words in one accepted style or the other.

Types of structures:

- Sentence Patterns
- Phrase Patterns
- Formulas
- Idioms

Difference between Statistical Approach and Structural Approach:

SR. NO.	**STATISTICAL APPROACH**	**STRUCTURAL APPROACH**
1	Statistical decision theory.	Human perception and cognition.
2	Quantitative features.	Morphological primitives
3	Fixed number of features.	Variable number of primitives.
4	Ignores feature relationships.	Captures primitives' relationships.
5	Semantics from feature position.	Semantics from primitives encoding.
6	Statistical classifiers.	Syntactic grammars.

Check your Progress-3

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the chapter.

vi. Define Pattern.

…………………………………………………………………………

……………………………………………………………..................

…………………………………………………………………………

…………………………………………………………………………

vii. Write your understanding about testing and training set.

…………………………………………………………………………

…………………………………………………………………………

……………………………………………………………..................

…………………………………………………………………………

viii. Define Classifier.

…………………………………………………………………….....

……………………………………………………………..................

…………………………………………………………………………

…………………………………………………………………….....

END UNIT EXERCISE

1. Discuss about ANN.
2. What are all the types of ANN?
3. Explain about Machine Learning in ANN.
4. State Bayes' Theorem with Example.
5. Briefly explain about Bayesian Network.
6. Write the Semantics of Bayesian Network.
7. What is learning?
8. Discuss about the various forms of learning.
9. Explain EBL.
10. Describe Machine Learning in AI.
11. Briefly discuss about Pattern Recognition with example.
12. Write Applications of Pattern Recognition.
13. Discuss about the design and principles of Pattern Recognition.

Answer to Check Your Progress:

i. A computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.

ii. Causal Component and Actual numbers.

iii. To understand the network as the representation of the Joint probability distribution and to understand the network as an encoding of a collection of conditional independence statements.

iv. Learning is the activity of gaining knowledge or skill by studying, practicing, being taught, or experiencing something. Learning enhances the awareness of the subjects of the study.

v. Explanation-based learning (EBL) deals with an idea of single-example learning.

vi. **Pattern** is everything around in this digital world. A pattern can either be seen physically or it can be observed mathematically by applying algorithms.

vii. Training data is a certain percentage of an overall dataset along with testing set.

viii. **Classifier** is used to partition the feature space into class-labeled decision regions.

SUGGESTED READINGS:

1. Donald Tveter, “The Pattern Recognition of Basis of Artificial Intelligence”, Wiley, 1998.
2. Christopher M. Bishop, “Pattern Recognition and Machine Learning” , Springer, 2006.
3. https://www.tutorialride.com/artificial-intelligence/learning-and-expert-system-in-ai.htm
4. https://certes.co.uk/types-of-artificial-intelligence-a-detailed-guide/
5. https://certes.co.uk/types-of-artificial-intelligence-a-detailed-guide/
6. https://www.geeksforgeeks.org/pattern-recognition-introduction/

UNIT - VII Expert Systems

Structure

7.1 Introduction

Expert Systems (ES) are one of the noticeable research areas in the field of Artificial Intelligence (AI). The researchers in Computer Science Department at Stanford University introduced the expert systems for the first time. The expert systems are designed specially to solve real-time problems.

Expert Systems

Artificial Intelligence is a piece of software that simulates the behavior and judgment of a human or an organization that has experts in a particular domain is known as an expert system. It does by acquiring relevant knowledge from its knowledge base and interpreting it according to the user's problem. The data in the knowledge base is added by humans that are expert in a particular domain and this software is used by a non-expert user to acquire some information. It is widely used in many areas such as medical diagnosis, accounting, coding, games etc. An expert system is an AI software that uses knowledge stored in a knowledge base to solve problems that would usually require a human expert thus preserving a human expert's knowledge in its knowledge base. They can advise users as well as provide explanations to them about how they reached a particular conclusion or advice.

Examples: There are many examples of expert system. Some of them are given below:

- **MYCIN:** One of the earliest expert systems based on backward chaining. It can identify various bacteria that can cause severe infections and can also recommend drugs based on the person's weight.
- **DENDRAL:** It was an artificial intelligence based expert system used for chemical analysis. It used a substance's spectrographic data to predict it's molecular structure.
- **R1/XCON:** It could select specific software to generate a computer system wished by the user.
- **PXDES:** It could easily determine the type and the degree of lung cancer in a patient based on the data.
- **CaDet:** It is a clinical support system that could identify cancer in its early stages in patients.
- **DXplain:** It was also a clinical support system that could suggest a variety of diseases based on the findings of the doctor.

Expert systems (ES) are one of the prominent research domains of AI. It is introduced by the researchers at Stanford University, Computer Science Department.

The expert systems are the computer applications developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence and expertise.

Capabilities of Expert Systems

The expert systems are capable of −

- Advising
- Instructing and assisting human in decision making
- Demonstrating
- Deriving a solution
- Diagnosing
- Explaining
- Interpreting input
- Predicting results
- Justifying the conclusion
- Suggesting alternative options to a problem

They are incapable of −

- Substituting human decision makers
- Possessing human capabilities
- Producing accurate output for inadequate knowledge base
- Refining their own knowledge

Expert Systems Limitations

No technology can offer easy and complete solution. Large systems are costly, require significant development time, and computer resources. ESs have their limitations which include −

- Limitations of the technology
- Difficult knowledge acquisition
- ES are difficult to maintain
- High development costs

Expert System Technology

There are several levels of ES technologies available. Expert systems technologies include −

- **Expert System Development Environment** − The ES development environment includes hardware and tools. They are −
  - Workstations, minicomputers, mainframes.
  - High level Symbolic Programming Languages such as **LIS**t **P**rogramming (LISP) and **PRO**grammation en **LOG**ique (PROLOG).
  - Large databases.

- **Tools** − They reduce the effort and cost involved in developing an expert system to large extent.
    - Powerful editors and debugging tools with multi-windows.
    - They provide rapid prototyping
    - Have Inbuilt definitions of model, knowledge representation, and inference design.
- **Shells** − A shell is nothing but an expert system without knowledge base. A shell provides the developers with knowledge acquisition, inference engine, user interface, and explanation facility. For example, few shells are given below −
    - Java Expert System Shell (JESS) that provides fully developed Java API for creating an expert system.
    - *Vidwan*, a shell developed at the National Centre for Software Technology, Mumbai in 1993. It enables knowledge encoding in the form of IF-THEN rules.

Development of Expert Systems: General Steps

The process of ES development is iterative. Steps in developing the ES include −

Identify Problem Domain

- The problem must be suitable for an expert system to solve it.
- Find the experts in task domain for the ES project.
- Establish cost-effectiveness of the system.

Design the System

- Identify the ES Technology
- Know and establish the degree of integration with the other systems and databases.
- Realize how the concepts can represent the domain knowledge best.

Develop the Prototype

From Knowledge Base: The knowledge engineer works to −

- Acquire domain knowledge from the expert.
- Represent it in the form of If-THEN-ELSE rules.

Test and Refine the Prototype

- The knowledge engineer uses sample cases to test the prototype for any deficiencies in performance.
- End users test the prototypes of the ES.

Develop and Complete the ES

- Test and ensure the interaction of the ES with all elements of its environment, including end users, databases, and other information systems.
- Document the ES project well.
- Train the user to use ES.

Maintain the System

- Keep the knowledge base up-to-date by regular review and update.
- Cater for new interfaces with other information systems, as those systems evolve.

Benefits of Expert Systems

- **Availability** − They are easily available due to mass production of software.
- **Less Production Cost** − Production cost is reasonable. This makes them affordable.
- **Speed** − They offer great speed. They reduce the amount of work an individual puts in.
- **Less Error Rate** − Error rate is low as compared to human errors.
- **Reducing Risk** − They can work in the environment dangerous to humans.
- **Steady response** − They work steadily without getting motional, tensed or fatigued.

An expert system is a computer program that uses artificial intelligence (AI) technologies to simulate the judgment and behavior of a human or an organization that has expert knowledge and experience in a particular field.

Typically, an expert system incorporates a knowledge base containing accumulated experience and an inference or rules engine -- a set of rules for applying the knowledge base to each particular situation that is described to the program. The system's capabilities can be enhanced with additions to the knowledge base or to the set of rules. Current systems may include machine learning capabilities that allow them to improve their performance based on experience, just as humans do.

The concept of expert systems was first developed in the 1970s by Edward Feigenbaum, professor and founder of the Knowledge Systems Laboratory at Stanford University. Feigenbaum explained that the world was moving from data processing to "knowledge processing," a transition which was being enabled by new processor technology and computer architectures.

Expert systems have played a large role in many industries including in financial services, telecommunications, healthcare, customer service, transportation, video games, manufacturing, aviation and written communication.

Two early expert systems broke ground in the healthcare space for medical diagnoses: Dendral, which helped chemists identify organic molecules, and MYCIN, which helped to identify bacteria such as bacteremia and meningitis, and to recommend antibiotics and dosages.

A more recently developed expert system, ROSS, is an artificially-intelligent attorney based on IBM's Watson cognitive computing system. ROSS relies on self-learning systems that use data mining, pattern recognition, deep learning and natural language processing to mimic the way the human brain works.

Expert systems and AI systems have evolved so far that they have spurred debate about the fate of humanity in the face of such intelligence, with authors such as Nick Bostrom, professor of philosophy at Oxford University, pondering if computing power has surpassed our ability to control it.

What is an expert system :

An expert system is a computer program or we can say an application that can solve complex of the complex problem in a particular domain. It is designed using the concept of Artificial Intelligence and was first introduced in the Department of Computer Science, Stanford University. **The expert system can perform at the extraordinary level of human intelligence or human experts.** In this article, we will discuss the various components of an expert system with the diagram.

Basically, the expert system represents the knowledge of the human expert in the form of heuristic. It can be also considered as an instance of a decision support system. The knowledge base and decision rule are the most unique and distinguishing features of an expert system.

The concept of the expert system is normally based on assumption that an expert’s knowledge can be stored in computer memory and then applied by other when needed. An expert system shares knowledge of a human expert in a specific area of study such as production engineering, genetic engineering and so on. It is found that the problem-solving capabilities of an expert system are as good as that of human experts or sometimes even better than the human experts.

An Expert System is defined as an interactive and reliable computer-based decision-making system which uses both facts and heuristics to solve complex decision-making problems. It is considered at the highest level of human intelligence and expertise. It is a computer application which solves the most complex issues in a specific domain.

What is an Expert System?

An Expert System is defined as an interactive and reliable computer-based decision-making system which uses both facts and heuristics to solve complex decision-making problems. It is considered at the highest level of human intelligence and expertise. It is a computer application which solves the most complex issues in a specific domain.

The expert system can resolve many issues which generally would require a human expert. It is based on knowledge acquired from an expert. It is also capable of expressing and reasoning about some domain of knowledge. Expert systems were the predecessor of the current day artificial intelligence, deep learning and machine learning systems.

Examples of Expert Systems

Following are examples of Expert Systems

- **MYCIN:** It was based on backward chaining and could identify various bacteria that could cause acute infections. It could also recommend drugs based on the patient's weight.
- **DENDRAL:** Expert system used for chemical analysis to predict molecular structure.
- **PXDES:** Expert system used to predict the degree and type of lung cancer
- **CaDet:** Expert system that could identify cancer at early stages
-

Participant	**Role**
Domain Expert	He is a person or group whose expertise and knowledge is taken to develop an expert system.
Knowledge Engineer	Knowledge engineer is a technical person who integrates knowledge into computer systems.
End User	It is a person or group of people who are using the expert system to get to get advice which will not be provided by the expert.

End User	It is a person or group of people who are using the expert system to get to get advice which will not be provided by the expert.

The process of Building An Expert Systems

- Determining the characteristics of the problem
- Knowledge engineer and domain expert work in coherence to define the problem
- The knowledge engineer translates the knowledge into a computer-understandable language. He designs an inference engine, a reasoning structure, which can use knowledge when needed.
- Knowledge Expert also determines how to integrate the use of uncertain knowledge in the reasoning process and what type of explanation would be useful.

Conventional System vs. Expert system

Conventional System	**Expert System**
Knowledge and processing are combined in one unit.	Knowledge database and the processing mechanism are two separate components.
The programme does not make errors (Unless error in programming).	The Expert System may make a mistake.
The system is operational only when fully developed.	The expert system is optimized on an ongoing basis and can be launched with a small number of rules.
Step by step execution according to fixed algorithms is required.	Execution is done logically & heuristically.
It needs full information.	It can be functional with sufficient or insufficient information.

ES Building Tools on the Market

Traditionally, ES tools have been categorized by their hardware platform: PC- or Macintosh-based, workstation-based, or mainframe-based. (For example, see Harmon 1992a).

Recently, new types of tools have come on the market that are characterized according to tasks (e.g., diagnosis, planning) and problem-solving approaches (e.g., case-based reasoning or model based reasoning). These second generation tools encode the problem-solving know-how gained through building applications in different areas using the first generation tools. The emergence of such tools reflects the market condition in which vertical tools are perceived to be easier to use and easier to sell. A problem-specific, or task-specific, tool contains knowledge representation schemes and reasoning methods found useful for a particular class of applications and a task ontology associated with the problem class.

Table 3.1 lists most of the commercial tools developed in Japan. They are broadly categorized as general purpose, task-specific, solution-specific, and development methodology tools (i.e., tools for training implementors in the methodology for developing expert systems). There are also more general-purpose tools on the market than the list might indicate. A general purpose tool such as ES/KERNEL represents a class of tools, with a version of the tool for different types of hardware platforms -- ES/KERNEL/W for workstations; ES/KERNEL/H for mainframes and super-computers; ES/KERNEL/P for the personal computers; and ES/KERNEL/D for on-line data processing.

In addition to the tools developed by the Japanese, foreign-made tools, primarily American, make up about 30 percent of the tools used in fielded expert systems in Japan. This JTEC panel is not aware of any Japanese tools being sold in the U.S. Components of the next version of ES/KERNEL -- ES/KERNEL2 -- are being developed in Europe and will be marketed there. Figure 3.2 shows the relative popularity of the more common tools in use. The four most popular tools are those developed by domestic computer manufacturers, Hitachi, Fujitsu, and NEC.

Human Expert	Artificial Expertise
Perishable	Permanent
Difficult to Transfer	Transferable
Difficult to Document	Easy to Document
Unpredictable	Consistent
Expensive	Cost effective System

Benefits of expert systems

- It improves the decision quality
- Cuts the expense of consulting experts for problem-solving
- It provides fast and efficient solutions to problems in a narrow area of specialization.
- It can gather scarce expertise and used it efficiently.
- Offers consistent answer for the repetitive problem
- Maintains a significant level of information
- Helps you to get fast and accurate answers
- A proper explanation of decision making
- Ability to solve complex and challenging issues
- Expert Systems can work steadily work without getting emotional, tensed or fatigued.

Limitations of the expert system

- Unable to make a creative response in an extraordinary situation
- Errors in the knowledge base can lead to wrong decision
- The maintenance cost of an expert system is too expensive
- Each problem is different therefore the solution from a human expert can also be different and more creative

Over the past several years there have been many implementations of expert systems using various tools and various hardware platforms, from powerful LISP machine workstations to smaller personal computers.

The technology has left the confines of the academic world and has spread through many commercial institutions. People wanting to explore the technology and experiment with it have a bewildering selection of tools from which to choose. .

There continues to be a debate as to whether or not it is best to write expert systems using a high-level shell, an AI language such as LISP or Prolog, or a conventional language such as C.

This book is designed to teach you how to build expert systems from the inside out. It presents the various features used in expert systems, shows how to implement them in Prolog, and how to use them to solve problems.

The code presented in this book is a foundation from which many types of expert systems can be built. It can be modified and tuned for particular applications. It can be used for rapid prototyping. It can be used as an educational laboratory for experimenting with expert system concepts.

Expert Systems

Expert systems are computer applications which embody some non-algorithmic expertise for solving certain types of problems. For example, expert systems are used in diagnostic applications servicing both people and machinery. They also play chess, make financial planning decisions, configure computers, monitor real time systems, underwrite insurance policies, and perform many other services which previously required human expertise.

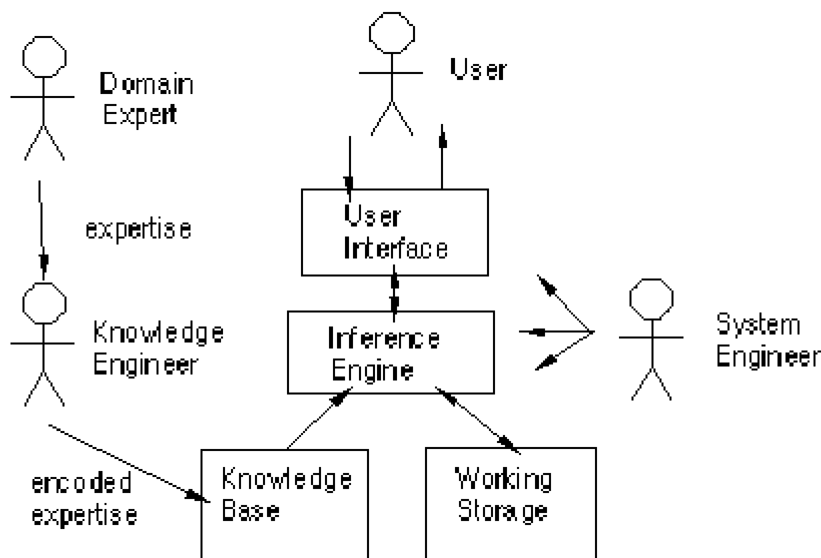


Figure 7.1 Expert system components and human interfaces

Expert systems have a number of major system components and interface with individuals in various roles. These are illustrated in figure 1.1. The major components are:

- Knowledge base - a declarative representation of the expertise, often in IF THEN rules;

- Working storage - the data which is specific to a problem being solved;

- Inference engine - the code at the core of the system which derives recommendations from the knowledge base and problem-specific data in working storage;

- User interface - the code that controls the dialog between the user and the system.

To understand expert system design, it is also necessary to understand the major roles of individuals who interact with the system. These are:

- Domain expert - the individual or individuals who currently are experts solving the problems the system is intended to solve;

- Knowledge engineer - the individual who encodes the expert's knowledge in a declarative form that can be used by the expert system;

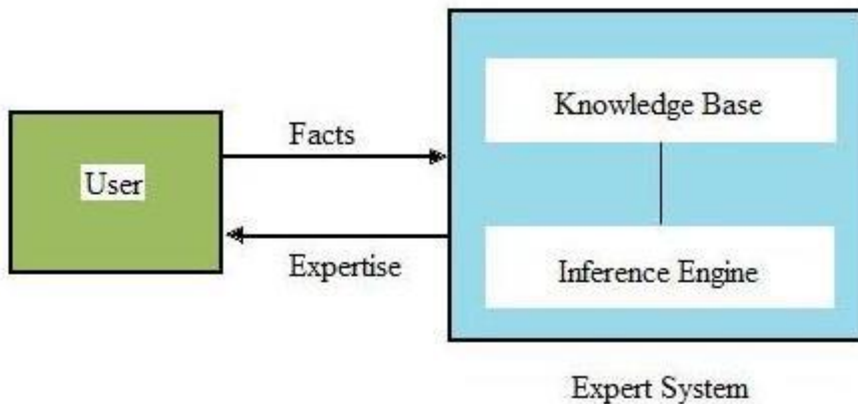
- User - the individual who will be consulting with the system to get advice which would have been provided by the expert.

Many expert systems are built with products called expert system shells. The shell is a piece of software which contains the user interface, a format for declarative knowledge in the knowledge base, and an inference engine. The knowledge engineer uses the shell to build a system for a particular problem domain.

Expert systems are also built with shells that are custom developed for particular applications. In this case there is another key individual:

- System engineer - the individual who builds the user interface, designs the declarative format of the knowledge base, and implements the inference engine.

Depending on the size of the project, the knowledge engineer and the system engineer might be the same person. For a custom built system, the design of the format of the knowledge base, and the coding of the domain knowledge are closely related. The format has a significant effect on the coding of the knowledge.



One of the major bottlenecks in building expert systems is the knowledge engineering process. The coding of the expertise into the declarative rule format can be a difficult and tedious task. One major advantage of a customized shell is that the format of the knowledge base can be designed to facilitate the knowledge engineering process.

The objective of this design process is to reduce the semantic gap. Semantic gap refers to the difference between the natural representation of some knowledge and the programmatic representation of that knowledge. For example, compare the semantic gap between a mathematical formula and its representation in both assembler and FORTRAN. FORTRAN code (for formulas) has a smaller semantic gap and is therefor easier to work with.

Since the major bottleneck in expert system development is the building of the knowledge base, it stands to reason that the semantic gap between the expert's representation of the knowledge and the representation in the knowledge base should be minimized. With a customized system, the system engineer can implement a knowledge base whose structures are as close as possible to those used by the domain expert.

This book concentrates primarily on the techniques used by the system engineer and knowledge engineer to design customized systems. It explains the various types of inference engines and knowledge bases that can be designed, and how to build and use them. It tells how they can be mixed together for some problems, and customized to meet the needs of a given application.

7.2 Definition

The expert systems are systems which are having astonishing human skill and intelligence. These systems are built using computer applications. The development of these systems mainly focuses on solving complex problems in real-time. This system solves problem by IF-THEN rules.

7.3 Components of Expert Systems

The three main components of Expert Systems are,

- The Knowledge Base
- Inference Engine and
- User Interface

However, there will be a human expert to feed knowledge to the knowledge base.

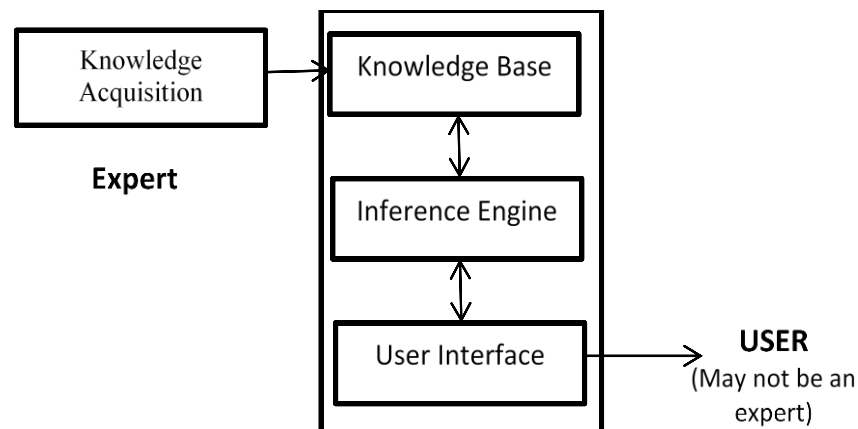


Figure : Components of Expert System

A domain expert is responsible for providing knowledge to the knowledge base. In this case the expert will be a knowledge engineer who is capable of serving knowledge. Since this process is time consuming and acquires a lot of skill from the expert, it mostly limits the functioning and designing of the expert system in commercial environment. An individual expert who is having ability to create, modify or add any changes to the knowledge base will use the knowledge base program.

Components of an expert system:

- **Knowledge base:** The knowledge base represents facts and rules. It consists of knowledge in a particular domain as well as rules to solve a problem, procedures and intrinsic data relevant to the domain.
- **Inference engine:** The function of the inference engine is to fetch the relevant knowledge from the knowledge base, interpret it and to find a solution relevant to the user's problem. The inference engine acquires the rules from its knowledge base and applies them to the known facts to infer new facts. Inference engines can also include an explanation and debugging abilities.
- **Knowledge acquisition and learning module:** The function of this component is to allow the expert system to acquire more and more knowledge from various sources and store it in the knowledge base.
- **User interface:** This module makes it possible for a non-expert user to interact with the expert system and find a solution to the problem.
- **Explanation module:** This module helps the expert system to give the user an explanation about how the expert system reached a particular conclusion.

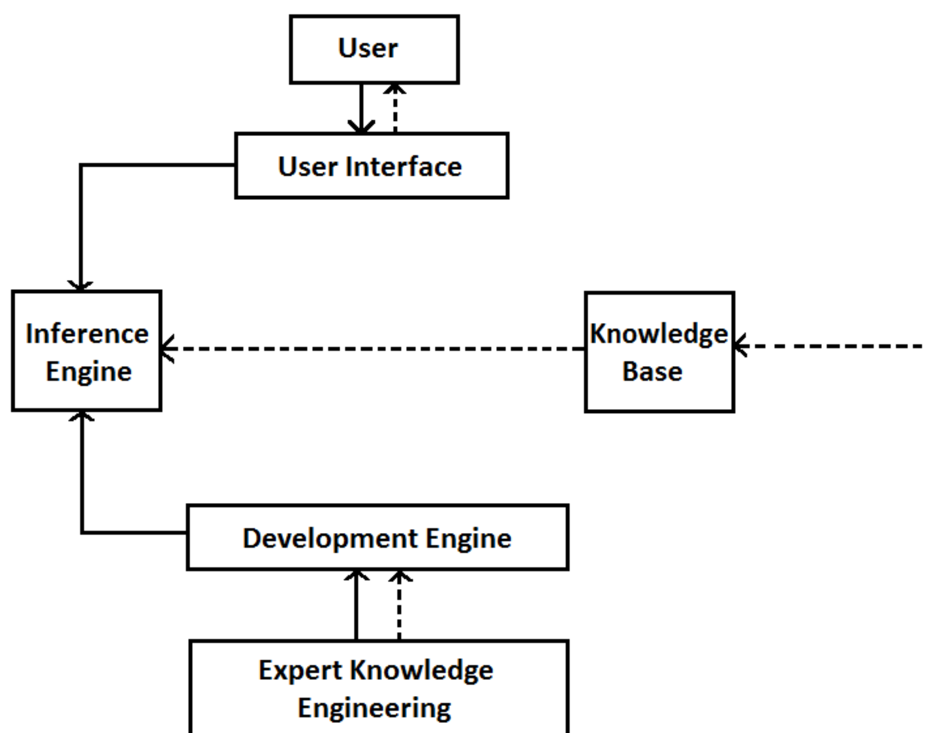


Figure: Components

7.3.1 Knowledge Base

It contains domain-specific and high-quality knowledge.

Knowledge is required to exhibit intelligence. The success of any ES majorly depends upon the collection of highly accurate and precise knowledge.

What is Knowledge?

The data is collection of facts. The information is organized as data and facts about the task domain. **Data, information,** and **past experience** combined together are termed as knowledge.

It contains domain-specific and high-quality knowledge.

Knowledge is required to exhibit intelligence. The success of any ES majorly depends upon the collection of highly accurate and precise knowledge.

What is Knowledge?

The data is collection of facts. The information is organized as data and facts about the task domain. **Data, information,** and **past experience** combined together are termed as knowledge.

Components of Knowledge Base

The knowledge base of an ES is a store of both, factual and heuristic knowledge.

- **Factual Knowledge** − It is the information widely accepted by the Knowledge Engineers and scholars in the task domain.
- **Heuristic Knowledge** − It is about practice, accurate judgement, one’s ability of evaluation, and guessing.

This is the first and important component of an expert system which contains domain specific knowledge. This knowledge is what actually needed to exhibit intelligence. It is the component which holds the expert’s problem-solving knowledge.

It is the place where the knowledge given by the expert is stored. It contains facts, information, guideline, descriptions and past experience etc., are grouped together to form knowledge.

The knowledge base provides all the information that are must for understanding, formulating and solving a problem. The knowledge stores both the heuristic and factual knowledge.

- Factual or truthful knowledge – as the name indicates it is the truthful information which is accepted by Scholars and Knowledge experts in most cases.
- Heuristic or empirical knowledge – it deals with training, correct judgement and one's ability of estimating and evaluating.

Knowledge representation

It is the method used to organize and formalize the knowledge in the knowledge base. It is in the form of IF-THEN-ELSE rules.

Knowledge Acquisition

The success of any expert system majorly depends on the quality, completeness, and accuracy of the information stored in the knowledge base.

The knowledge base is formed by readings from various experts, scholars, and the **Knowledge Engineers**. The knowledge engineer is a person with the qualities of empathy, quick learning, and case analyzing skills.

He acquires information from subject expert by recording, interviewing, and observing him at work, etc. He then categorizes and organizes the information in a meaningful way, in the form of IF-THEN-ELSE rules, to be used by interference machine. The knowledge engineer also monitors the development of the ES.

It contains the fact that describes the problem area and knowledge representation technique that describes manner. That means the knowledge base contains a really high-quality and extraordinary knowledge in that particular domain. The term problem domain is used to describe the problem. Or basically, we can say that the knowledge base is the set of rules. The rules in the knowledge base are usually coded in the form- if x, then y where x is a condition, y is an action to be taken if the condition is true.

7.3.2 Inference Engine

Use of efficient procedures and rules by the Inference Engine is essential in deducting a correct, flawless solution.

In case of knowledge-based ES, the Inference Engine acquires and manipulates the knowledge from the knowledge base to arrive at a particular solution.

In case of rule based ES, it −

- Applies rules repeatedly to the facts, which are obtained from earlier rule application.
- Adds new knowledge into the knowledge base if required.
- Resolves rules conflict when multiple rules are applicable to a particular case.

To recommend a solution, the Inference Engine uses the following strategies −

- Forward Chaining
- Backward Chaining

The inference engine is the rule for defining how an expert's method interprets the information in a suitable manner. The inference engine takes the information from the knowledge base and processes it to get a solution. So, it is considered as a brain in expert system. This engine can work either in forward or backward chaining.

The Inference Engine uses two strategies –

1) Forward chaining

   This strategy answers the question, "What can occur next?". Forward chaining can make multiple passes. Forward chaining is suitable when the aim is to find some conclusions from the information given in knowledge base.

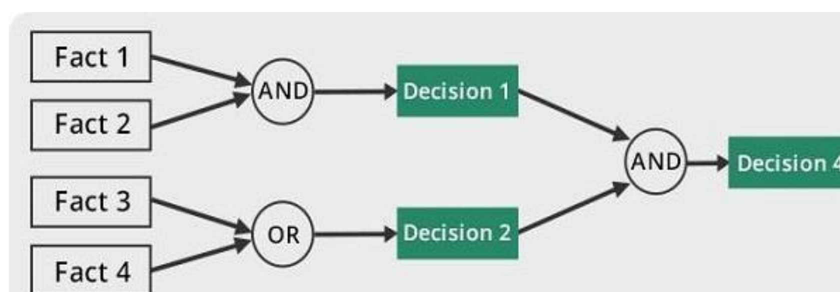


Figure : Forward Chaining Diagram

2. Backward chaining

With this strategy, an expert system finds out the answer to the question, **“Why this happened?”**

On the basis of what has already happened, the Inference Engine tries to find out which conditions could have happened in the past for this result. This strategy is followed for finding out cause or reason. For example, diagnosis of blood cancer in humans.

This strategy answers the question, “Why this occured?”. Backward chaining is suitable when there are multiple rules and several goal variables.

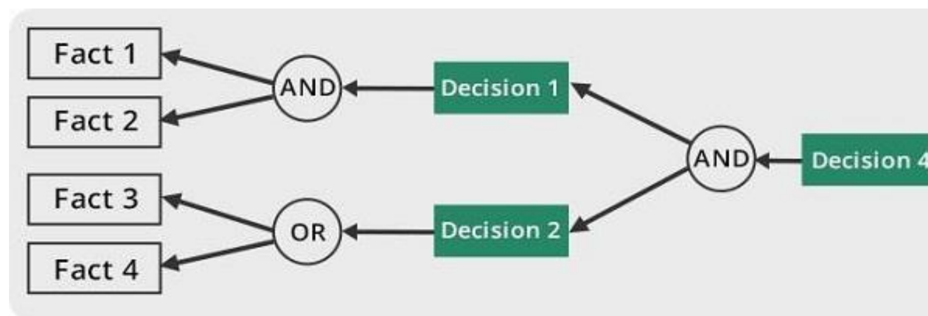


Figure : Backward Chaining

The inference engine is one of the most important components of an expert system. The inference engine of the expert system is the rule that defines how the expert process interprets the knowledge in an appropriate manner. The inference engine work in either forward chaining or backward chaining.

In simple, the inference engine takes the knowledge base and then it applies processing to it. The inference engine processes a massive amount of data in some kind of consistent way and it comes out with a conclusion. It works as a brain in an expert system.

Backward chaining process faster than the forward chaining because it doesn’t make multiple passes through the rule set. Backward chaining is especially appropriate when-

1. There are multiple goal variables.
2. There are many rules.
3. All or most of the rules don’t have examined in the process of reaching the solution.

7.3.3 User Interface

This is a component which provides a communication between expert system and the user of the expert system. It is usually a Natural language processing task, which could be easily understood by the user. The user may or may not be a domain expert in AI.

The user interface explains how the expert system leads to a specific recommendation. The explanation may be in the form of,

- Listing of rule numbers on the screen
- Verbal narrations in natural language
- Natural language displayed on the screen

It is used to communicate with the user. It is basically keep input from user for better intelligency and observe all basic human requirements and usable thoes in future.

The user interface is the most crucial part of the expert system. This component takes the user's query in a readable form and passes it to the inference engine. After that, it displays the results to the user. In other words, it's an interface that helps the user communicate with the expert system.

- **User interface:** This module makes it possible for a non-expert user to interact with the expert system and find a solution to the problem.
- **User interface** – that portion of the code which creates an easy to use system;
- • User interface - the code that controls the dialog between the user and the system.

User interface - that portion of the code which creates an easy to use system;

The acceptability of an expert system depends to a great extent on the quality of the user interface. The easiest to implement interfaces communicate with the user through a scrolling dialog as illustrated in figure 1.4. The user can enter commands, and respond to questions. The system responds to commands, and asks questions during the inferencing process.

More advanced interfaces make heavy use of pop-up menus, windows, mice, and similar techniques as shown in figure 1.5. If the machine supports it, graphics can also be a powerful tool for communicating with the user. This is especially true for the development interface which is used by the knowledge engineer in building the system.

```
Start of Bird Identification

what is color?
>white
what is size?
>large
....
The bird is a laysan_albatross
```

Figure Scrolling dialog user interface

User interface provides interaction between user of the ES and the ES itself. It is generally Natural Language Processing so as to be used by the user who is well-versed in the task domain. The user of the ES need not be necessarily an expert in Artificial Intelligence.

It explains how the ES has arrived at a particular recommendation. The explanation may appear in the following forms −

- Natural language displayed on screen.
- Verbal narrations in natural language.
- Listing of rule numbers displayed on the screen.

The user interface makes it easy to trace the credibility of the deductions.

It enables the users to enter instruction and information into the expert system and to receive information from it. The information is in the form of values assigned to certain variables. The user interface has two parts –

1. **Expert System Input:** A user can use method for input command, natural language and customize the interface.
2. **Expert System Output:** Expert systems are designed to provide output or solution for a specific domain.

Requirements of Efficient ES User Interface

- It should help users to accomplish their goals in shortest possible way.
- It should be designed to work for user's existing or desired work practices.
- Its technology should be adaptable to user's requirements; not the other way round.
- It should make efficient use of user input.

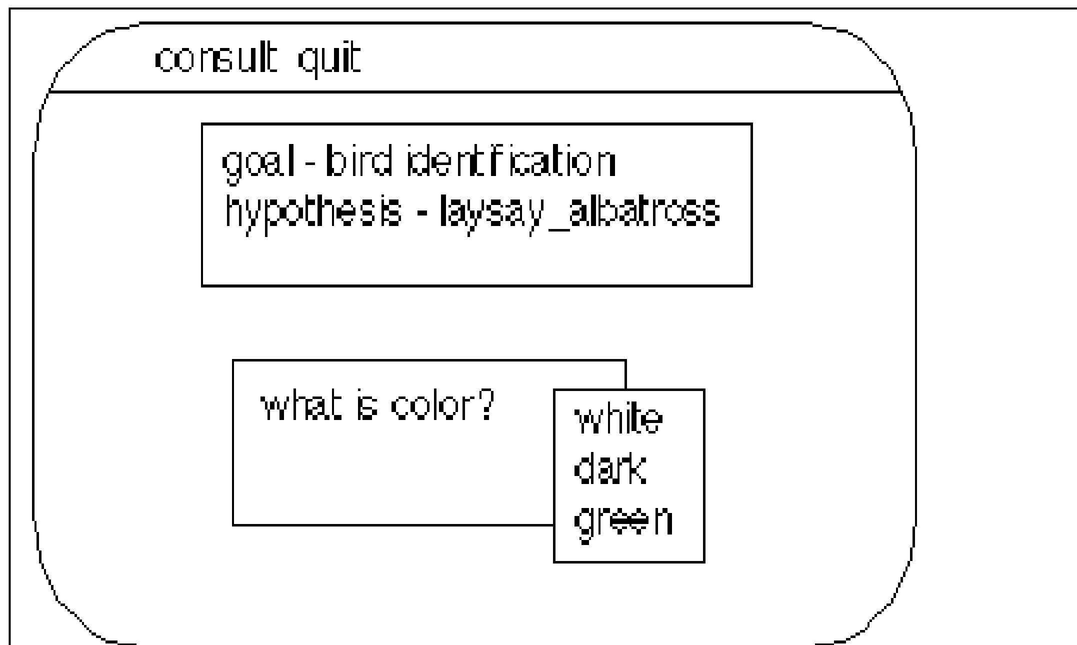


Figure : Window and menu user interface

Explanations

One of the more interesting features of expert systems is their ability to explain themselves. Given that the system knows which rules were used during the inference process, it is possible for the system to provide those rules to the user as a means for explaining the results.

This type of explanation can be very dramatic for some systems such as the bird identification system. It could report that it knew the bird was a black footed albatross because it knew it was dark colored and an albatross. It could similarly justify how it knew it was an albatross.

At other times, however, the explanations are relatively useless to the user. This is because the rules of an expert system typically represent empirical knowledge, and not a deep understanding of the problem domain. For example a car diagnostic system has rules which relate symptoms to problems, but no rules which describe why those symptoms are related to those problems.

Explanations are always of extreme value to the knowledge engineer. They are the program traces for knowledge bases. By looking at explanations the knowledge engineer can see how the system is behaving, and how the rules and data are interacting. This is an invaluable diagnostic tool during development.

7.4 Characteristic Features of Expert System

There is a tremendous increase in the growth of an expert system. Many new applications are updated by the continuing growth of expert system. An expert system generally aids the decision-making process. The expert system is a very interactive model that responses to queries, asks for doubts and makes suggestions. The main focus of an expert system is to give expert advice and suggestions to solve real time problems. Below are some of the important characteristics of expert system.

7.4.1 Domain Specific

Expert systems are used to solve domain specific problems. For example, in the medical field a diagnostic expert system should make all the essential data manipulation as a human expert would do. In making such a system, a developer should limit his scope of the system to just what is needed solve the target problem.

7.4.2 High level Performance

The performance of expert system must be high in order to achieve complex tasks better than which an ordinary system would do.

7.4.3 Reliability

An expert system must be capable of handling a problem like which a human expert would handle. In such case an expert system must be as consistent as human expert.

7.4.4 Understandable

The expert system should be capable of understanding and explaining the steps of reasoning while executing. Like to the human experts, the expert system should have an explanation capability.

7.4.5 Better Responsive

The expert system must be developed in such a way that it could give solution to a problem with minimal amount of time. The system should be able to take less time when compared with human expert.

7.4.6 Symbolic Representations

The expert system must be able to do symbolic computations as like natural language. This will better help in representation of rules.

7.4.7 Expertise knowledge

As like human expert, the expert system must be capable of applying its knowledge to get better solutions efficiently.

7.4.8 Justified Reasoning

If the user asks any justification for the solution given by expert system, it must be able to provide it. Usually the expert system provides the users, all the facts it used to attain its answer.

7.4.9 Explaining Capability

Expert system must be capable of explaining how a specific solution was given to a particular problem. This task is very crucial since it gives the user a chance to understand the system's logical ability.

7.4.10 Special Programming Languages

The expert system uses special languages like PROLOG and LISP which simplifies the coding process. These languages provide ease of adding, removing or substituting of new rules. It is also capable of managing memory. The use of special languages will help in optimization of the system, incremental compilation and in efficient search procedures.

Check your Progress

Note a:Write your answers in the space given below

b. Compare your notes with those given at the end of the unit

1. Explain Components of Expert System

………………………………………………………………………………………

………………………………………………………………………………………

Characteristics of an Expert System

Separates knowledge from control (related article)

Possesses expert knowledge

- expertise - extensive, task-specific knowledge; includes:

- facts (about problem area)

- theories (about problem area)

- hard and fast rules about the general problem area ('i' before 'e' except after 'c')

- heuristics of what to do in a given problem situation

- global strategies for solving these types of problems

- meta-knowledge (knowledge about knowledge)

- expert - fuzzy definition;

nonexperts outnumber experts in a field by 100 to 1

characteristics of an expert: adept at

- recognizing and formulating problem (once you have a good representation for a problem, it is almost solved)

- solving the problem quickly and properly

- explaining the solution

- learning from experience

- restructuring knowledge

- breaking rules

- determining relevance

- degrading gracefully

focuses expertise

reasons with symbols (which is what all computation is) - can relate to original problems of machine translation

reasons heuristically (cf. expertise above)

permits inexact reasoning (not essential)

Characteristics of Expert Systems

High performance: They should perform at the level of a human expert.

Adequate response time: They should have the ability to respond in a reasonable amount of time. Time is crucial especially for real time systems.

Reliability: They must be reliable and should not crash.

Understandable: They should not be a black box instead it should be able explain the steps of the reasoning process. It should justify its conclusions in the same way a human expert explains why he arrived at particular conclusion.

Major Characteristics of an Expert System

The Structure of an Expert System

The established definition of an "expert system" relies on the main abilities ascribed to human "experts", viz., they have specific expertise in some area(s). This expertise can be categorized as having "domain knowledge" and some ability (innate, learned, etc.) to apply that knowledge to solve problems within the domain.

An important realization in the design of software soltions: techniques employed by "automatic devices" (computer hardware, software, mechanical devices, automobile-airbags, expert systems) provide *simplifications* of possible solutions to a *simplification* of the problem. These simplifications are necessary because the real process may

- be too complex
- not be solvable
- not be understood
- take too long
- not be routinely reproducible
- etc.

The human model (of intelligent behavior) is partially explainable by:

- Short term memory
- Long term memory
- Reasoning facility

The computerized model follows a similar design:

Expert System	Traditional Program
Working memory	Variables
Knowledge Base	Files
Inference Engine	Program logic

Definition: knowledge base - the part of expert system that contains the domain (and possibly other types of) knowledge.

Definition: working memory - facts newly discovered (by inference from the knowledge base and other working memory contents; or from input to system.)

Definition: inference engine - algorithm which uses knowledge base and working memory to infer (items are then added to the working memory, or sometimes to the knowledge base.)

Check your Progress

Note a:Write your answers in the space given below

b. Compare your notes with those given at the end of the unit

1. List out the characteristics features of Expert System

………………………………………………………………………………………

………………………………………………………………………………………

Expert System Features

There are a number of features which are commonly used in expert systems. Some shells provide most of these features, and others just a few. Customized shells provide the features which are best suited for the particular problem. The major features covered in this book are:

- Goal driven reasoning or backward chaining - an inference technique which uses IF THEN rules to repetitively break a goal into smaller sub-goals which are easier to prove;

- Coping with uncertainty - the ability of the system to reason with rules and data which are not precisely known;

- Data driven reasoning or forward chaining - an inference technique which uses IF THEN rules to deduce a problem solution from initial data;

- Data representation - the way in which the problem specific data in the system is stored and accessed;

- User interface - that portion of the code which creates an easy to use system;

- Explanations - the ability of the system to explain the reasoning process that it used to reach a recommendation.

Goal-Driven Reasoning

Goal-driven reasoning, or backward chaining, is an efficient way to solve problems that can be modelled as "structured selection" problems. That is, the aim of the system is to pick the best choice from many enumerated possibilities. For example, an identification problem falls in this category. Diagnostic systems also fit this model, since the aim of the system is to pick the correct diagnosis.

The knowledge is structured in rules which describe how each of the possibilities might be selected. The rule breaks the problem into sub-problems. For example, the following top level rules are in a system which identifies birds.

IF
family is albatross and
color is white

THEN
bird is laysan albatross.

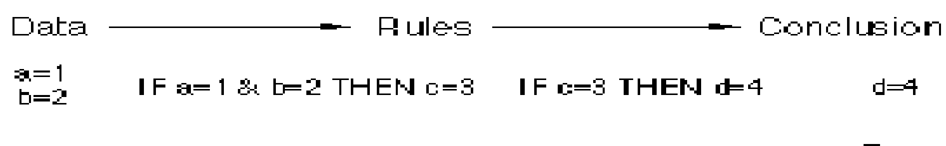
IF
family is albatross and
color is dark

THEN
bird is black footed albatross.

The system would try all of the rules which gave information satisfying the goal of identifying the bird. Each would trigger sub-goals. In the case of these two rules, the sub-goals of determining the family and the color would be pursued. The following rule is one that satisfies the family sub-goal:

IF
order is tubenose and
size large and
wings long narrow

THEN
family is albatross.



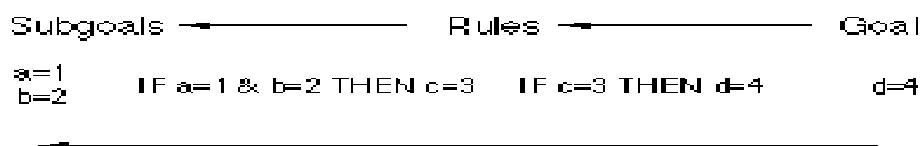


Figure 1.2. Difference between forward and backward chaining

The sub-goals of determining color, size, and wings would be satisfied by asking the user. By having the lowest level sub-goal satisfied or denied by the user, the system effectively carries on a dialog with the user. The user sees the system asking questions and responding to answers as it attempts to find the rule which correctly identifies the bird.

Uncertainty

Often times in structured selection problems the final answer is not known with complete certainty. The expert's rules might be vague, and the user might be unsure of answers to questions. This can be easily seen in medical diagnostic systems where the expert is not able to be definite about the relationship between symptoms and diseases. In fact, the doctor might offer multiple possible diagnoses.

For expert systems to work in the real world they must also be able to deal with uncertainty. One of the simplest schemes is to associate a numeric value with each piece of information in the system. The numeric value represents the certainty with which the information is known. There are numerous ways in which these numbers can be defined, and how they are combined during the inference process.

Data Driven Reasoning

For many problems it is not possible to enumerate all of the possible answers before hand and have the system select the correct one. For example, configuration problems fall in this category. These systems might put components in a computer, design circuit boards, or lay out office space. Since the inputs vary and can be combined in an almost infinite number of ways, the goal driven approach will not work.

The data driven approach, or forward chaining, uses rules similar to those used for backward chaining, however, the inference process is different. The system keeps track of the current state of problem solution and looks for rules which will move that state closer to a final solution.

A system to layout living room furniture would begin with a problem state consisting of a number of unplaced pieces of furniture. Various rules would be responsible for placing the furniture in the room, thus changing the problem state. When all of the furniture was placed, the system would be finished, and the output would be the final state. Here is a rule from such a system which places the television opposite the couch.

IF
unplaced tv and
couch on wall(X) and
wall(Y) opposite wall(X)

THEN
place tv on wall(Y).

This rule would take a problem state with an unplaced television and transform it to a state that had the television placed on the opposite wall from the couch. Since the television is now placed, this rule will not fire again. Other rules for other furniture will fire until the furniture arrangement task is finished.

Note that for a data driven system, the system must be initially populated with data, in contrast to the goal driven system which gathers data as it needs it. Figure 1.2 illustrates the difference between forward and backward chaining systems for two simplified rules. The forward chaining system starts with the data of **a=1** and **b=2** and uses the rules to derive **d=4**. The backward chaining system starts with the goal of finding a value for **d** and uses the two rules to reduce that to the problem of finding values for **a** and **b**.

Attribute-Value Pairs

color - white

Object-Attribute-Value Triples

arm_chair - width - 3
straight_chair - width - 2

Records

chairs

object	width	color	type
chair#1	3	orange	easy
chair#2	2	brown	straight

Frames

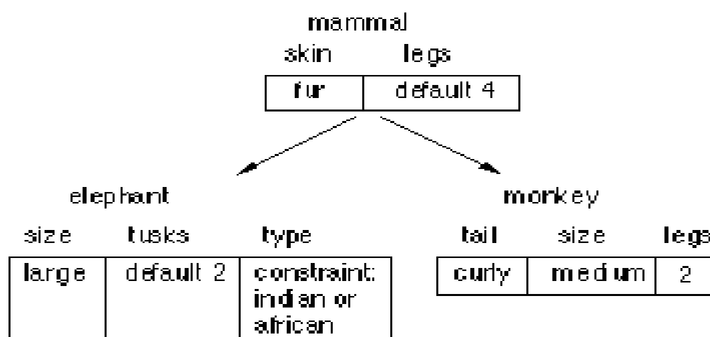


Figure 1.3. Four levels of data representation

Data Representation

For all rule based systems, the rules refer to data. The data representation can be simple or complex, depending on the problem. The four levels described in this section are illustrated in figure 1.3.

The most fundamental scheme uses attribute-value pairs as seen in the rules for identifying birds. Examples are color-white, and size-large.

When a system is reasoning about multiple objects, it is necessary to include the object as well as the attribute-value. For example the furniture placement system might be dealing with multiple chairs with different attributes, such as size. The data representation in this case must include the object.

Once there are objects in the system, they each might have multiple attributes. This leads to a record-based structure where a single data item in working storage contains an object name and all of its associated attribute-value pairs.

Frames are a more complex way of storing objects and their attribute-values. Frames add intelligence to the data representation, and allow objects to inherit values from other objects. Furthermore, each of the attributes can have associated with it procedures (called demons) which are executed when the attribute is asked for, or updated.

In a furniture placement system each piece of furniture can inherit default values for length. When the piece is placed, demons are activated which automatically adjust the available space where the item was placed.

Sample Applications

In chapters 2 through 9, some simple expert systems are used as examples to illustrate the features and how they apply to different problems. These include a bird identification system, a car diagnostic system, and a system which places furniture in a living room.

Chapters 10 and 11 focus on some actual systems used in commercial environments. These were based on the principles in the book, and use some of the code from the book.

The final chapter describes a specialized expert system which solves Rubik's cube and does not use any of the formalized techniques presented earlier in the book. It illustrates how to customize a system for a highly specialized problem domain.

Prolog

The details of building expert systems are illustrated in this book through the use of Prolog code. There is a small semantic gap between Prolog code and the logical specification of a program. This means the description of a section of code, and the code are relatively similar. Because of the small semantic gap, the code examples are shorter and more concise than they might be with another language.

The expressiveness of Prolog is due to three major features of the language: rule-based programming, built-in pattern matching, and backtracking execution. The rule-based programming allows the program code to be written in a form which is more declarative than procedural. This is made possible by the built-in pattern matching and backtracking which automatically provide for the flow of control in the program. Together these features make it possible to elegantly implement many types of expert systems.

There are also arguments in favor of using conventional languages, such as C, for building expert system shells. Usually these arguments center around issues of portability, performance, and developer experience. As newer versions of commercial Prologs have increased sophistication, portability, and performance, the advantages of C over Prolog decrease. However, there will always be a need for expert system tools in other languages. (One mainframe expert system shell is written entirely in COBOL.)

For those seeking to build systems in other languages, this book is still of value. Since the Prolog code is close to the logical specification of a program, it can be used as the basis for implementation in another language.

Assumptions

This book is written with the assumption that the reader understands Prolog programming. If not, *Programming in Prolog* by Clocksin and Mellish from Springer-Verlag is the classic Prolog text. *APT - The Active Prolog Tutor* by the author and published by Solution Systems in South Weymouth, Massachusetts is an interactive PC based tutorial that includes a practice Prolog interpreter.

An in depth understanding of expert systems is not required, but the reader will probably find it useful to explore other texts. In particular since this book focuses on system engineering, readings in knowledge engineering would provide complementary information. Some good books in this area are: *Building Expert Systems* by Hayes-Roth, Waterman, and Lenat; *Rule-Based Expert Systems* by Buchanan and Shortliffe; and *Programming Expert Systems in OPS5* by Brownston, Kant, Farrell, and Martin.

The Advantages of Using Expert System

Expert system has been reliably used in the business world to gain tactical advantages and forecast the market's condition. In this globalization era where every decision made in the business world is critical for success, the assistance provided from an expert system is undoubtedly essential and highly reliable for an organization to succeed. Examples given below will be the advantages for the implementation of an expert system in business:

1. **Providing consistent solutions** – It can provide consistent answers for repetitive decisions, processes and tasks. As long as the rule base in the system remains the same, regardless of how many times similar problems are being tested, the final conclusions drawn will remain the same.
2. **Provides reasonable explanations** – It has the ability to clarify the reasons why the conclusion was drawn and be why it is considered as the most logical choice among other alternatives. If there are any doubts in concluding a certain problem, it will prompt some questions for users to answer in order to process the logical conclusion.
3. **Overcome human limitations** – It does not have human limitations and can work around the clock continuously. Users will be able to frequently use it in seeking solutions. The knowledge of experts is an invaluable asset for the company. It can store the knowledge and use it as long as the organization needs.
4. **Easy to adapt to new conditions** – Unlike humans who often have troubles in adapting in new environments, an expert system has high adaptability and can meet new requirements in a short period of time. It also can capture new knowledge from an expert and use it as inference rules to solve new problems.

The Disadvantages of Using Expert System

Examples given below will be the disadvantages for the implementation of an expert system in business:

1. **Lacks common sense** – It lacks common sense needed in some decision making since all the decisions made are based on the inference rules set in the system. It also cannot make creative and innovative responses as human experts would in unusual circumstances.
2. **High implementation and maintenance cost** – The implementation of an expert system in business will be a financial burden for smaller organizations since it has high development cost as well as the subsequent recurring costs to upgrade the system to adapt in new environment.
3. **Difficulty in creating inference rules** – Domain experts will not be able to always explain their logic and reasoning needed for the knowledge engineering process. Hence, the task of codifying out the knowledge is highly complex and may require high
4. **May provide wrong solutions** – It is not error-free. There may be errors occurred in the processing due to some logic mistakes made in the knowledge base, which it will then provide the wrong solutions.

7.5 Unit – End Exercise

1. List down the components of expert system
2. Give some of the important characteristics features of Expert System

7.6 Answers to Check Your Progress

1. The three main components of Expert Systems are,
- The Knowledge Base
- Inference Engine and
- User Interface

2. Domain Specific: Expert systems are used to solve domain specific problems
   High level Performance: The performance of expert system must be high.
   Reliability: An expert system must be capable of handling a problem like which a human expert.
   Understandable: The expert system should be capable of understanding and explaining the steps of reasoning while executing. Like to the human experts, the expert system should have an explanation capability.
   Better Responsive: The system should be able to take less time when compared with human expert.

7.7 Suggested Readings

1. Introduction to Expert Systems, Jackson P., 3rd edition, Addison Wesley, ISBN 0-201-87686-8
2. Giarratano J., Riley G., Expert Systems, Principles and Programming, PWS Publishing Company, Boston., ISBN 0-534-93744-6
3. http://intelligence.worldofcomputing.net/ai-branches/expert-systems.html#.XanG-2bhVPY
4. http://intelligence.worldofcomputing.net/ai-branches/expert-systems.html#.XanG_mbhVPY
5. http://www.cs.oswego.edu/~odendahl/coursework/isc320/notes/jackson/01-ab-characteristics.html
6. https://www.brainkart.com/article/Characteristics-of-an-Expert-System_8929/

UNIT - VIII Rule Based System Architecture

Structure

8.1 Introduction

Rule-based system is a system that is used to collect, store and manipulate knowledge to infer information. These systems are also known as production systems since it produces information from the knowledge gained. Here the knowledge will be encoded as production rules. The rule-based system consists of both the IF-THEN rules, where left hand side is conditional part and left-hand side is conclusion part.

For example,

If: condition1 and condition2

Then: Take action5

From the above example it is clearly known that if the conditions satisfy the IF part on left side, then the action on the right side will be taken place. In rule-based system each rule is a small piece of knowledge which is given to the domain expertise.

In the rule-based system the rules or facts are given as input to the inference module. The information in inference module is stored in a separate database. The output generated from the inference module is provided to user interface.

In a rule-based system, the inference engine usually goes through a simple recognize-assert cycle. The control scheme is called forward chaining for data-driven reasoning, and backward chaining for goal-driven reasoning. The basic idea of forward chaining is when the premises of a rule (the **if** portion) are satisfied by the data, the expert system asserts the conclusions of the rule (the **then** portion) as true.

A **forward-chaining reasoning system** starts by placing initial data in its working memory. Then the system goes through a cycle of matching the premises of rules with the facts in the working memory, selecting one rule, and placing its conclusion in the working memory. This inference process is useful in searching for a goal or an interpretation, given a set of data. For example, XCON is a forward-chaining rule-based system (McDermott, 1982) that contains several thousand rules for designing configurations of computer components for individual customers. It was one of the first clear commercial successes of expert systems. Its underlying technology has been implemented in the general-purpose language OPS-5.

In a **backward-chaining reasoning system** the goal is initially placed in the working memory. The system matches rule conclusions with the goal, selects one rule, and places its premises in the working memory. The process iterates, with these premises becoming new goals to match against rule conclusions. Thus, the system works backward from the original goal until all the subgoals in the working memory are known to be true.

Subgoals may also be solved by asking the user for information. For example, MYCIN's inference engine uses a backward-chaining control strategy. From its goal of finding significant disease-causing organisms, MYCIN uses its rules to reason backward to the data available. Once it finds such organisms, it attempts to select a therapy to treat the disease(s). Since it was designed as a consultant for physicians, MYCIN was given the ability to explain both its reasoning and its knowledge (Buchanan and Shortliffe, 1984).

Given a fixed reasoning method, the process of searching through alternative solutions can be affected through the structuring and ordering of the rules in implementations. For example, in production systems, a rule of the form **"if *p* and *q* and r then *s*"** may be interpreted in backward chaining as a procedure of four steps: **to do *s*, first do *p*, then do *q*, then do *r*.** Although the procedural interpretation of rules reduces the advantages of declarative representation, it can be used to reflect more efficient heuristic solution strategies. For instance, the premises of a rule may be ordered so that the one that is most likely to fail or is easiest to be satisfied will be tried first.

To illustrate forward and backward chaining, consider a simple example with the following rules in the knowledge base:

Rule 1: **if a and b, then c**

Rule 2: **if c and d, then p (s1)**

Rule 3: **if not d and not e, then p (s2) or p (s3)**

Rule 4: **if not d and e, then p (s4**)

The symbols *a* to *e* and *s*1 to *s*4 represents objects, and *p* represents a property of objects.

To perform backward-chaining reasoning, the top-level goal, *p*(*X*), is placed in the working memory as shown in Figure 5.3(A), where *X* is a variable that can match with any object. The conclusions of three rules (rules 2, 3, and 4) match with the expression in the working memory. If we solve conflicts in favor of the lower-numbered rule, then rule 2 will be selected and fire. This causes *X* to be bound to s1 and the two premises of rule 2 to be placed in the working memory as in Figure 5.3(B). Then, since the conclusion of rule 1 matches with a fact in the working memory, we then fire rule 1 and place its promises in the working memory as in Figure 5.3(C). At this point, there are three entries in the working memory (*a*, *b*, *d*) that do not match with any rule conclusion. The expert system will query the user directly about these subgoals. If the user confirms them as true, the expert system will have successfully determined the causes for the top-level goal *p*(*X*).

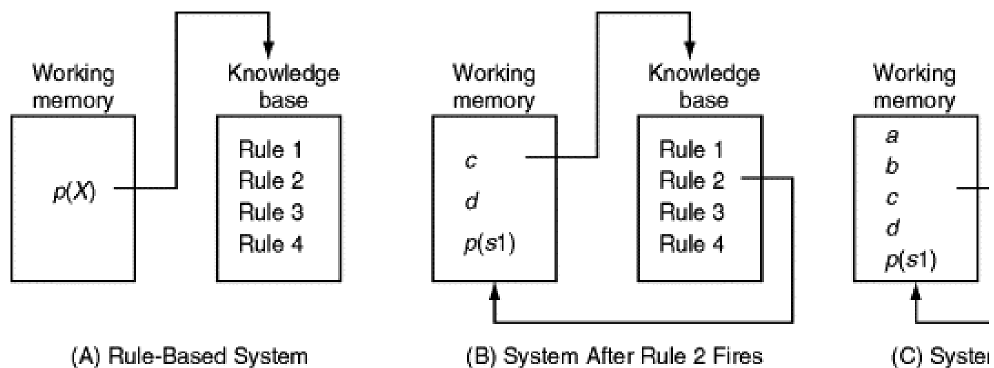


FIGURE The Rule-Based System Throughout a Goal-Driven Inference Process

The control of the previous backward-chaining process performs a depth-first search, in which each new subgoal is searched exhaustively first before moving onto old subgoals. Other search strategies, such as breadth-first search, can also be applied.

Given the same set of rules, forward chaining can also be applied to derive new conclusions from given data. For example, the algorithm of forward chaining with breadth-first search is as follows: Compare the content of the working memory with the premises of each rule in the rule base using the ordering of the rules. If the data in the working memory match a rule's premises, the conclusion is placed in the working memory, and the control moves to the next rule. Once all rules have been considered, the control starts again from the beginning of the rule sets.

Fundamental and Practical Aspects of Neural Computing

D.R. Baughman, Y.A. Liu, in Neural Networks in Bioprocessing and Chemical Engineering, 1995

1 Fuzzy-Logic Systems

Fuzzy logic grew out of a desire to quantify rule-based systems. Rule-based reasoning is grounded in qualitative knowledge representation, and fuzzy logic allows us to mesh a quantitative approach with the qualitative representation. It provides a way to quantify certain qualifiers such as *approximately, often, rarely, several, few,* and *very.* Figure 2.33 shows the relationship of fuzzy-logic systems to the two main areas of artificial intelligence (expert systems and neural networks) based on knowledge type and information framework. The knowledge type is divided into structured (based on rules) and unstructured, and the information framework is divided into symbolic and numerical, as described in Section 1.1A.

Information framework

Knowledge type	symbolic	numerical
structured	expert system	fuzzy-logic system
unstructured	————	neural network

Sign in to download full-size image

Figure 2.33. Relationship of fuzzy-logic systems to expert systems and neural networks

(Kosko, 1992).

a Representation of Fuzzy-Logic Variables

In this section, we adopt and update part of the discussion on fuzzy-logic systems from Quantrille and Liu (1991, pp. 208-210). First, note that fuzzy logic is not a substitute for statistics. Instead, we use fuzzy logic only when statistical reasoning is inappropriate. Statistics expresses the extent of knowledge (or the lack thereof) about a value, and it relies on tools such as variance, standard deviation, and confidence intervals. Fuzzy logic, on the other hand, expresses the absence of *a sharp boundary* between sets of information. For example, using fuzzy logic, we may write:

- Crude oil fractionation is clearly an energy-intensive unit operation, 1.0.
- Thermal cracking is a very energy-intensive unit operation, 0.9.
- Catalytic reforming is a somewhat energy-intensive unit operation, 0.6.
- Catalytic cracking is an energy-intensive unit operation, 0.3.
- Open-air evaporation of brine to produce salt is not an energy-intensive unit operation, 0.0.

Here, the fuzzy logic delineates the lack of a sharp boundary between *clearly* energy-intensive (1.0) and *not at all* energy-intensive (0.0). Crude fractionation is very energy-intensive, while open-air evaporation of brine is not at all energy-intensive. Thermal cracking, catalytic reforming, and catalytic cracking cannot be considered either very energy-intensive or non-energy-intensive. Thus, fuzzy logic does not quantify the lack of knowledge in a statistical sense. Instead, it quantifies *the degree or extent* of certain words and boundaries between sets of information.

To use fuzzy logic, we first need *a fuzzy set.* In a fuzzy set, the transition from membership to non-membership is not well-defined. We quantify *the degree of membership* with values between 0 (not a member) and 1 (definitely a member). Figure 2.34 shows a representation of energy requirement in fuzzy terms (very low, low, moderate, high, very high). The transition from one discrete segment to another (e.g., low to moderate) is not defined exactly. These regions overlap based on what one expert says is energy-intensive compared to what another says.

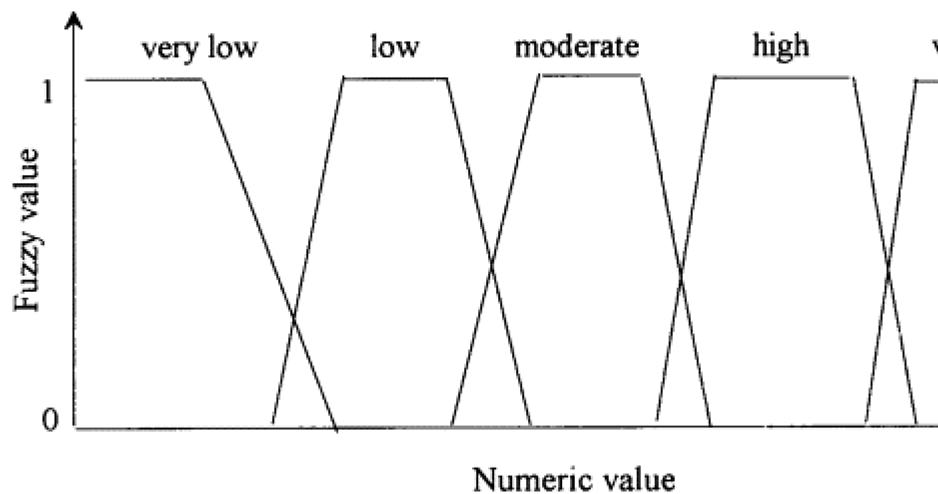


Figure Representation of energy requirement in fuzzy terms.

b Conversion between Numeric and Fuzzy-Logic Variables

This section describes how to convert a numeric variable to a fuzzy-logic variable through *a fuzzifier,* and to convert the fuzzy-logic variable back to a numeric variable through *a defuzzifier.*

We will use the low and moderate regions of the energy-requirement example (Figure 2.34) to demonstrate these two transformations. Figure 2.35 shows these regions of the energy requirement with numerical values given for the transition regions. Although we recommend using symmetric transition regions in which the sum of the member contributions equals 1 (e.g., low = 0.5 and moderate = 0.5) as in Figure 2.35, the transition regions can be staggered so that the fuzzy members do not total 1 (e.g., low = 0.5 and moderate = 0.3).

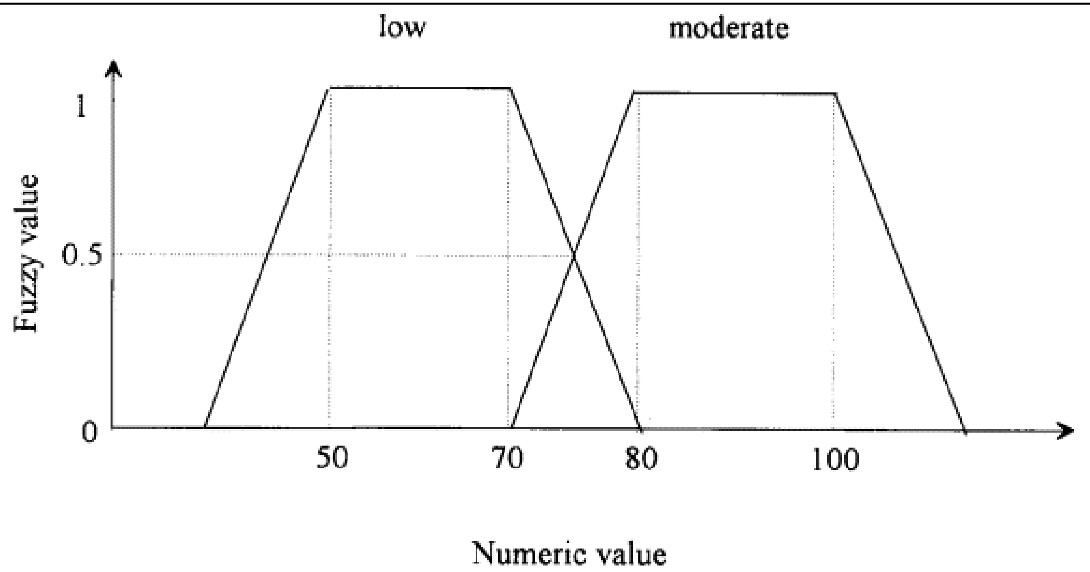


Figure 2.35. The low and moderate regions of the energy requirement represented in both numeric and fuzzy-logic variables.

In the following conversion examples, the numeric variable is denoted *numeric(x)* and the fuzzy-logic variables are denoted *fuzzy (very low, low, moderate, high, very high).* For numeric values of energy requirement that are definitely within a group (e.g., 50 to 70 for low, and 80 to 100 for moderate), we simply assign a value of 1 to the respective member of the group and 0 to the other members.

(2.60)numeric (60) = fuzzy (0,1,0,0,0)
(2.61)numeric (90) = fuzzy (0,0,1,0,0)

For numeric values in the transition regions, we use a linear interpolation between the beginning and ending values of that region:

(2.62)numeric (74) = fuzzy (0,80−7470−80, 70−7480−70,0,0) = fuzzy(0,.6,0.4,0,0)

The fuzzy-logic values in Equation 2.62 then represent the probability that the energy requirement is low (0.4) or moderate (0.6).

Similarly, we can convert the fuzzy-logic variable back to a numeric variable using the exact opposite process. As an example, we use a fuzzy-logic value of 0.7 for low and 0.3 for moderate.

(2.63)fuzzy (0,0.7,0.3,0,0) ⇒0.7 = 80−x70−80 or 0.3 = 70−x80−70 (x − 73)

c Union and Intersection of Fuzzy Sets

Fuzzy logic plays a critical role in developing expert networks (see Chapter 6) because of its ability to use *fuzzy reasoning.* To understand fuzzy reasoning, we need two important concepts of classical set theory frequently used in expert systems: *union* and *intersection.* These concepts allow us to combine related fuzzy sets of information.

With our energy-intensive unit-operation example, the fuzzy set is:

{crude oil fractionation (1.0), thermal cracking (0.9), catalytic reforming (0.6), catalytic cracking (0.3)}

The open-air evaporation of brine to produce salt has a degree of membership of 0.0, and therefore, is not a member of the set.

We can now apply the union and intersection operations to fuzzy sets too. Let us define two fuzzy sets:

I = {x1/ i1, x2/ i2, …,xn/ in}J = {x1/ j1, x2/ j2, …,xp/ jp}

where x_1, x_2 … are members of the set with nonzero degrees of membership i_1 i_2, … (for set I) and j_1 j_2, … (for set J). Note that the sets *do not* need to have the same number of members; set I has n members, and set J has p members.

The union of two fuzzy sets is the fuzzy set containing the members of each set with the maximum degree of membership of that element in either set:

I ∪ J = {x1/(max(i1,j1)), x2/(max(i2,j2)),…}

The intersection of two fuzzy sets is the fuzzy set containing the members of each set with the minimum degree of membership of that element in both sets:

I ∩ J = {x1/(max(i1,j1)), x2/(max(i2,j2)),…}

For example, we consider the two sets:

> I = {crude oil fractionation/1.0, thermal cracking/0.9, catalytic reforming/0.6, catalytic cracking/0.3}
>
> J = {crude oil fractionation/0.8, thermal cracking/0.75, catalytic reforming/0.7, catalytic cracking/0.2, polymerization/0.1}

I ∪ J = {crude oil fractionation/1.0, thermal cracking/0.9, catalytic reforming/0.7, catalytic cracking/0.3, polymerization/0.1}

I ∩ J = {crude oil fractionation/0.8, thermal cracking/0.75, catalytic reforming/0.6,catalytic cracking/0.2}

Davis and Gandikota (1990) discuss fuzzy sets in more detail, and we use their simple example here to demonstrate reasoning with fuzzy sets. If we have qualitative values for flow rate (F) and pressure (P) of a chemical process, we may write a rule defining an abnormal system:

The system is abnormal if:

1.

Both F and P are high, OR

2.

F is low, OR P is low.

Let us make F a fuzzy set of flow rates, and P a fuzzy set of pressures, with the following degrees of membership:

F = {low_F/0.5, high_F/0.3, normal_F/0.2)

P = {low_P/0.8, high_P/0.15, normal_P/0.05}

Now let us determine the following:

1.*Certainty of high_F and high_P:* determined by the intersection of fuzzy sets F and P. Thus, certainty is the minimum degree of membership of high_F and high_P: certainty = min(0.3,0.15) = 0.15.
2.*Certainty of low_F or low_P:* determined by the union of fuzzy sets F and P. Thus, certainty is the maximum degree of membership of low_F and low_P: certainty = max(0.5,0.8) = 0.8.
3.*Overall uncertainty:* determined by taking the maximum certainties of both results, i.e., certainty = max(0.15,0.8) = 0.8.

Note again that the certainty in these rules is *not* to be interpreted as some type of “confidence limit” in the conclusion drawn. Instead, the certainty represents confidence in the qualitative values of the flow rate and pressure in the fuzzy sets.

Rule-based (production) systems have a long history [10] and have been applied to a variety of applications. A rule-based system has a knowledge base represented as a collection of "rules" that are typically expressed as "if-then" clauses. The set of rules forms the knowledge base that is applied to the current set of facts. Rule-based systems provide a method for representing inferential knowledge by using a simple "if-then" form, which is relatively easy to state and understand. The rule paradigm is naturally understood by humans. The basic architecture of a production rule system is shown in Figure

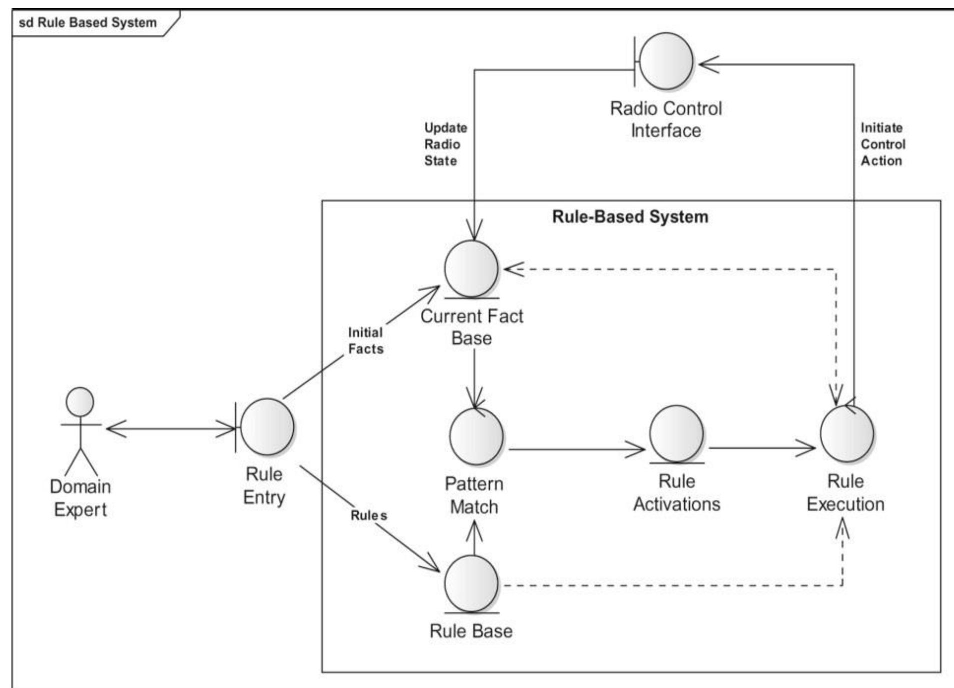


Figure Basic architecture of a rule-based system. The production rule matches the current state in working memory to one or more rules in the knowledge base.

As illustrated in the figure, the current state of the system is represented as a set of facts or assertions in working memory. The corpus of knowledge is stored within a set of rules that form the rule base. The inference engine performs a pattern match of the antecedents or conditional portion of the production rule against the set of assertions in the working memory. When there is a match, the rule is tagged for possible execution, building a set of rule activations. Because more than one rule may match the current state in working memory, some mechanism is usually provided for resolving the conflict and deciding which rule should be executed from the set of possible rule activations.

The selected rule is then executed, or "fired," resulting in some action being performed or one or more facts being asserted to (added) or retracted from (removed) the working memory and performing any control calls to the radio through the API. Any external data (e.g., radio state and sensory input) that are represented in working memory are updated. The updated statistical information in the current fact base is input into the pattern-matching process and the cycle continues.

Figure 12.9 illustrates a rule-based implementation. The syntax shown is a common form for production systems. The antecedents are expressed as a set of tuples in parentheses. The implied relationship between the tuples is usually AND. The use of the question mark (?) in front of a term (e.g., ?waveform) denotes that the item is a variable. So, any assertion in the fact base that matches the pattern results in the variable being set to the value in the fact base.

```
(defrule is-spectrum-available
      "Check current set of sensed frequencies for availability"
   (not (spectrum-sensed ?freq))
   (spectrum-requested ?freq ?waveform)
   =>
   (assert (spectrum-available ?freq)))

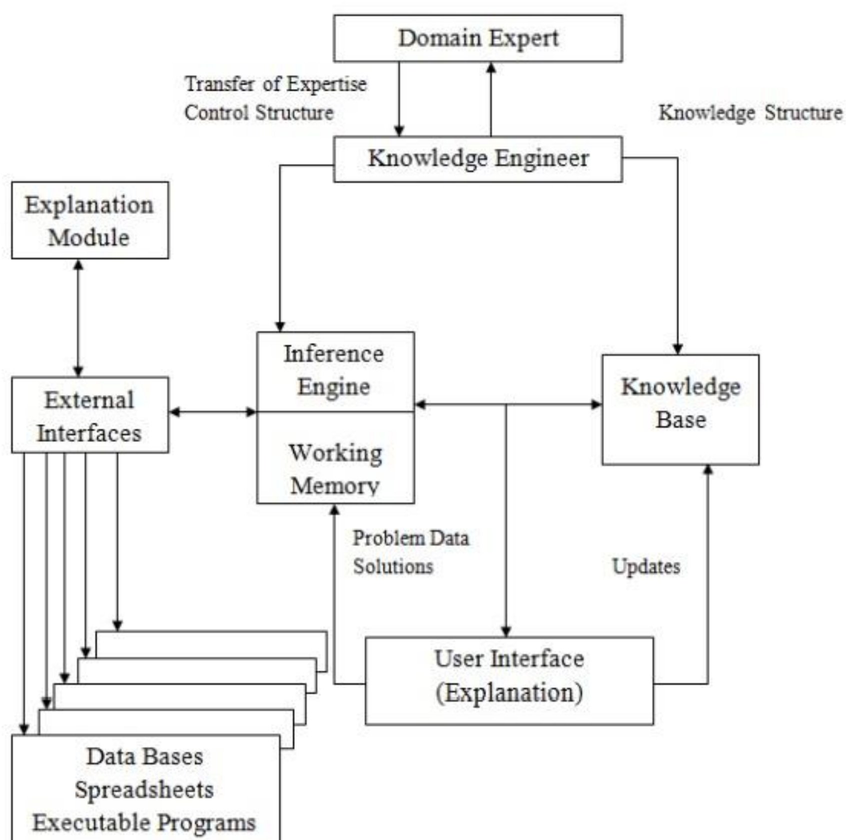
(defrule issue-start-waveform
      "Initialize state information to start waveform
      on the available frequency"
   ?fact1 <- (spectrum-requested ?freq ?waveform)
   ?fact2 <- (spectrum-available ?freq)
   =>
   (retract ?fact1)
   (retract ?fact2)
   ?wf-status <- (start-wf ?waveform)
   (assert (spectrum-in-use ?freq ?waveform))
   (assert (start-waveform-status ?wf-status)))
```

Figure. An example of rule-based reasoning.

There are two rules defined in the knowledge base: *is-spectrum-available* and *issue-waveform-start*. The first rule checks to see whether there is an existing request for a waveform to be started on a particular frequency, (spectrum-requested ?freq ?waveform), and that the frequency is available, (not (spectrum-sensed ?freq)). If so, it asserts, that is, adds to the fact base, that the request is allowable: (spectrum-available ?freq).

The second rule is activated when it has been confirmed that there is a specific frequency request by a waveform, (spectrum-requested ?freq ?waveform), and that the spectrum is available, (spectrum-available ?freq), which was asserted by the first rule. Both these assertions are remembered by the rule in temporary variables: ?fact1 and ?fact2. The rule retracts ?fact1 and ?fact2, removing them from the working memory, and then issues a call to start the waveform through an external function, **start-wf**.

One of the shortcomings of the rule-based paradigm, however, is that it typically has no means to introspect the knowledge within the system. In other words, the system's knowledge provides guidance regarding the domain of the system and responds based on the set of knowledge and the current state of the environment. However, the rule engine cannot scan through the rules to adjust them, add rules, or delete rules. To accomplish these activities, an additional set of reasoning must be implemented. Thus, the learning algorithm would be applied to monitor which rules were applied in a particular situation, assess the success of the decision process based on the actual outcome versus the predicted outcome, and then have at its disposal access to the rules in a form that the learning algorithm can understand and process.



8.2 User Interface

It is the place where the expert system and user interact with each other. this the communication module through which the user submits queries to the expert system. Usually the queries will be asked to get more clarification about the solution which is given to solve problems. The query will be in a language which an expert system can understand. The user interface interacts with the user in a natural language. The user interface also provides facilities such as graphical interfaces which can make the dialog user friendly.

. **User Interface:** It is the mechanism by which the user and the expert system communicate with each other i.e. the use interacts with the system through a user interface. It acts as a bridge between user and expert system. This module accepts the user queries and submits those to the expert system. The user normally consults the expert system for following reasons.

a) To get answer of his/her queries.

b) To get explanation about the solution for psychological satisfaction.

The user interface module is designed in such a way that at user level it accepts the query in a language understandable by expert system. To make the expert system user friendly, the user interface interacts with the user in natural language. The user interface provides as much facilities as possible such as menus, graphical interfaces etc. to make the dialog user friendly and more attractive.

8.3 Explanation Module

The explanation module explains the reasoning of the system to a user. It provides the user with an explanation of the reasoning process when requested. The credibility of expert system will be established only when it is able to explain "how and why" a particular conclusion is drawn. This explanation increases the belief of user in the expert system.

a) **Explanation(How):** To respond to a how query, the explanation module traces the chain of rules fired during a consolation with the user. This explanation mode can be activated once the process is over. It explains how a fact was deduced by the system and similarly how a rule was/wasn't used. The simplest way to specify this is to explain the rule which allows the deduction. For e.g.

If the system (S) will give information about the parent-child relationship to the user (U) then the followings can be possible.

S: My diagnosis is "A is the father of B"

U: How?

S: The result was obtained by the addition of following facts and rules.

Fact no 11: A is the parent of Hari.

Fact no 15: A is a male.

Fact no 110: X is father of Y:

X is parent of Y, X is male.

So A is the father of B.

b) **Explanation (Why)?** To respond to a why query, the explanation module must be able to explain why certain information is needed by the inference engine to complete a step in the reasoning process. This mode of explanation is dynamic and is used when the system functions in backward chaining. If the user doesn't understand the reason behind the question. The system is capable of explaining why the question was asked. For example S: Is the following true?

A is the father of B.

U:Why?

S:I need the fact:

A is the father of B to establish the following fact "B is the son of A".

By using the rule no. 4:

A is the father of B:

B is the son of A.

8.4 Working Memory

Working Memory

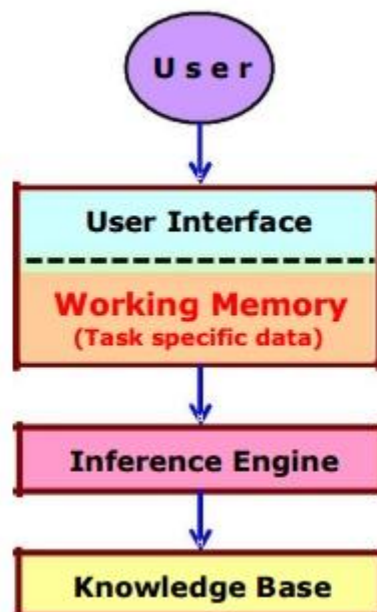
Working memory refers to task-specific data for a problem. The contents of the working memory, changes with each problem situation. Consequently, it is the most dynamic component of an expert system, assuming that it is kept current.

‡ Every problem in a domain has some unique data associated with it.

‡ Data may consist of the set of conditions leading to the problem, its parameters and so on.

‡ Data specific to the problem needs to be input by the user at the time of using, means consulting the expert system. The Working memory is related to user interface

Fig. below shows how Working memory is closely related to user interface of the expert system.



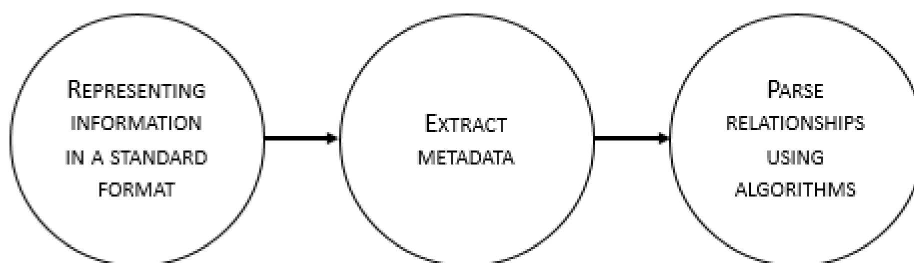
8.5 Knowledge Engineering

Knowledge engineering is one of the building blocks of artificial intelligence (AI). It attempts to emulate the judgment and behavior of a human with expertise in a field or domain. In this lesson, we'll define knowledge engineering and its processes, understand where it fits within the AI landscape, and provide real-world examples of its application.

What is Knowledge Engineering?

Imagine an education company wanting to automate the teaching of children in subjects from biology to computer science (requiring to capture the knowledge of teachers and subject matter experts) or Oncologists choosing the best treatment for their patients (requiring expertise and knowledge from information contained in medical journals, textbooks, and drug databases).

Knowledge Engineering is the process of imitating how a human expert in a specific domain would act and take decisions. It looks at the **metadata** (information about a data object that describes characteristics such as content, quality, and format), structure and processes that are the basis of how a decision is made or conclusion reached. Knowledge engineering attempts to take on challenges and solve problems that would usually require a high level of human expertise to solve. Figure 1 illustrates the knowledge engineering pipeline.



Knowledge Engineering Processes

In terms of its role in **artificial intelligence (AI)**, knowledge engineering is the process of understanding and then representing human knowledge in **data structures**, **semantic models** (conceptual diagram of the data as it relates to the real world) and **heuristics** (rules that lead to solution to every problem taken in AI). **Expert systems**, and **algorithms** are examples that form the basis of the representation and application of this knowledge.

General suggestions about the knowledge acquisition process are summarized in rough chronological order below:

1. Observe the person solving real problems.
2. Through discussions, identify the kinds of data, knowledge and procedures required to solve different types of problems.
3. Build scenarios with the expert that can be associated with different problem types.
4. Have the expert solve a series of problems verbally and ask the rationale behind each step.
5. Develop rules based on the interviews and solve the problems with them.
6. Have the expert review the rules and the general problem solving procedure.
7. Compare the responses of outside experts to a set of scenarios obtained from the project's expert and the ES.

Note that most of these procedures require a close working relationship between the knowledge engineer and the expert.

Practical Considerations

The preceding section provided an idealized version of how ES projects might be conducted. In most instances, the above suggestions are considered and modified to suit the particular project. The remainder of this section will describe a range of knowledge acquisition techniques that have been successfully used in the development of ES.

Operational Goals

After an evaluation of the problem domain shows that an ES solution is appropriate and feasible, then realistic goals for the project can be formulated. An ES's operational goals should define exactly what level of expertise its final product should be able to deliver, who the expected user is and how the product is to be delivered. If participants do not have a shared concept of the project's operational goals, knowledge acquisition is hampered.

Pre-training

Pre-training the knowledge engineer about the domain can be important. In the past, knowledge engineers have often been unfamiliar with the domain. As a result, the development process was greatly hindered. If a knowledge engineer has limited knowledge of the problem domain, then pre-training in the domain is very important and can significantly boost the early development of the ES.

Knowledge Document

Once development begins on the knowledge base, the process should be well

8.6 Knowledge Base

It is the set of production rules. In rule-based architecture the IF-THEN pairs are represented as rules, where IF part contains premises of rules and THEN part contains the action part. Knowledge base is the main portion of expert system. Here the knowledge is represented in a form of tree structure containing root frame and sub frames. A complex knowledge base can be designed on the basis of several frames and a simple knowledge base can have only one root frame.

Knowledge Representation is a way to transform human knowledge to machine understandable format.

Knowledge representation formalizes and organized the knowledge required to build an expert system.

A number of knowledge Representation techniques have been devised:

Production Rules:

A rule is a condition and an action associated with it. Aathe condition part is identified by keyword "IF". It lists a set of conditions in some logical combination. Actions are specified in "THEN" part.

As the IF part is satisfied: then THEN part actions can be taken. The piece of knowledge represented by the production rule is used to produce the line of reasoning.

As human thinking is evolved on the basis of situation-> conclusion or condition -> action basis, this Model is predominantly used representing knowledge in ES.

Check your Progress

Note a:Write your answers in the space given below
b. Compare your notes with those given at the end of the unit
1. Define Rule-Based System

Expert system is built around a knowledge base module.

Expert system contains a formal representation of the information provided by the domain expert. This information may be in the form of problem-solving rules, procedures, or data intrinsic to the domain. To incorporate these information into the system, it is necessary to make use of one or more knowledge representation methods. Some of these methods are described here.

Transferring knowledge from the human expert to a computer is often the most difficult part of building an expert system.

The knowledge acquired from the human expert must be encoded in such a way that it remains a faithful representation of what the expert knows, and it can be manipulated by a computer.

Three common methods of knowledge representation evolved over the years are IF-THEN rules, Semantic networks and Frames.

The first two methods were illustrated in the earlier lecture slides on knowledge representation therefore just mentioned here. The frame based representation is described more.

1. **IF-THEN rules**
2. If a1 , a2 , ... , an
3. Then b1 , b2 , ... , bn where
4. each ai is a condition or situation, and
5. each bi is an action or a conclusion.

2. Semantic Networks

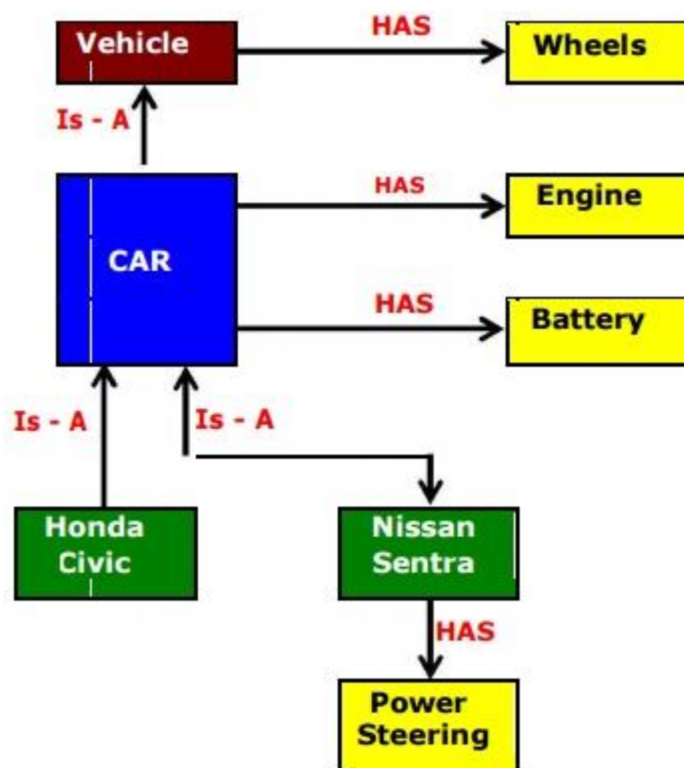
In this scheme, knowledge is represented in terms of objects and relationships between objects. The objects are denoted as nodes of a graph. The relationship between two objects are denoted as a link between the corresponding two nodes.

The most common form of semantic networks uses the links between nodes to represent **IS-A** and **HAS** relationships between objects.

Example of Semantic Network

The Fig. below shows a car IS-A vehicle; a vehicle HAS wheels.

This kind of relationship establishes an inheritance hy hiera in the network, with the objects lower down in the network inheriting properties from the objects higher up.



Semantic nets:

A semantic net are semantic network is a knowledge representation technique used for propositional information.

Here, the knowledge is represented as as objects and relationships between objects. They are two dimensional representations of knowledge.

Representation provide basic structure for organizing knowledge. It uses graphical notations to draw the networks.

Semantic nets consists of nodes, links and link lables. Objects are denoted by nodes of graph while links indicate relations among the objects.

Nodes appear as circles around ellipses or rectangles to represent objects such as physical objects, concepts or situations. Links are drawn as arrows to express the relationship between objects, and link labels specify specifications of relationships.

Relationships can be off two types "IS-A" or "HAS" relationship. IS-A relationship stands for on object being part of other related object. And HAS-A relationship indicated on object consists of the other related object t.

These relationships are nothing but superclass subclass relationships. It is assumed that all members of a subclass will inherit all the properties of their superclass. That's how semantic network allows efficient representations of inheritance reasoning.

Frames:

Frames provide a convenient structure for representing objects that are typical to stereotypical situations.

Frame is a type of schema used in many Artificial Intelligence applications including vision and natural language processing. Frames are also useful for representing common sense knowledge.

Frames can represent concepts, situations, attributes of concepts, relationships between concepts, and also procedures to explain their relationships. It allows nodes to have structures and hence is regarded as 3-D representations of knowledge.

A frame is also known as unit, schema, or list. Typically, a frame consists of a list of properties of the object and associated values for the properties; similar to the fields and values; also called as slots and slot filters. The contents of slot can be a

In this technique, knowledge is decomposed into highly modular pieces called frames, which are generalized record structures. Knowledge consist of concepts, situations, attributes of concepts, relationships between concepts, and procedures to handle relationships as well as attribute values.

‡ Each concept may be represented as a separate frame.

‡ The attributes, the relationships between concepts, and the procedures are allotted to slots in a frame.

‡ The contents of a slot may be of any data type - numbers, strings, functions or procedures and so on.

‡ The frames may be linked to other frames, providing the same kind of inheritance as that provided by a semantic network.

A frame-based representation is ideally suited for objected-oriented programming techniques. An example of Frame-based representation of knowledge is shown in next slide.

Example : Frame-based Representation of Knowledge. Two frames, their slots and the slots filled with data type are shown.

Frame	***Car***
Inheritance Slot	***Is-A***
Value	***Vehicle***
Attribute Slot	***Engine***
Value	***Vehicle***
Value	***1***
Value	
Attribute Slot	***Cylinders***
Value	***4***
Value	***6***
Value	***8***
Attribute Slot	***Doors***
Value	***2***
Value	***5***
Value	***4***

Frame	***Car***
Inheritance Slot	***Is-A***
Value	***Car***
Attribute Slot	***Make***
Value	***Honda***
Value	
Value	
Attribute Slot	***Year***
Value	***1989***
Value	
Value	
Attribute Slot	
Value	
Value	
Value	

8.7 Inference Engine

The inference engine replies to the user queries through I/O interface. It uses both the static knowledge and dynamic information. The inference engine module will get information from the knowledge base and applies it to find an answer to the problem. This inference engine makes inferring by analyzing which facts satisfies the rules. Then the satisfied rules will execute based on their priority. There are three stages in which the inferring process is carried out. They are match, select and execute.

The inference engine accepts user input queries and responses to questions through the I/O interface. It uses the dynamic information together with the static knowledge stored in the knowledge base. The knowledge in the knowledge base is used to derive conclusions about the current case as presented by the user's input. Inference engine is the module which finds an answer from the knowledge base. It applies the knowledge to find the solution of the problem. In general, inference engine makes inferences by deciding which rules are satisfied by facts, decides the priorities of the satisfied rules and executes the rule with the highest priority. Generally inferring process is carried out recursively in 3 stages like match, select and execute. During the match stage, the contents of working memory are compared to facts and rules contained in the knowledge base. When proper and consistent matches are found, the corresponding rules are placed in a conflict set.

Check your Progress

Note a:Write your answers in the space given below

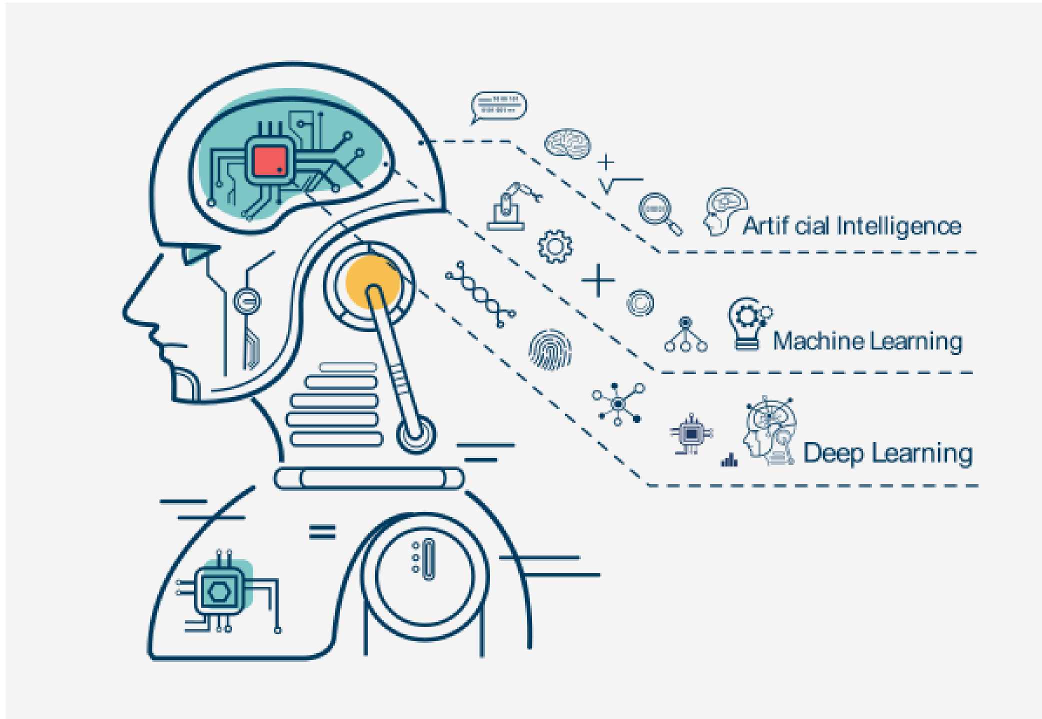
b. Compare your notes with those given at the end of the unit

1. Describe Explanation Module.

……………………………………………………………………………………

Artificial Intelligence (AI) is undoubtedly one of the main enablers of the digital transformation of modern enterprises. It is used as a vehicle for increased automation of business processes and as a means of optimizing enterprise decisions. **Building AI that works** means its applications transcend almost all sectors of the economy from finance, cybersecurity, IoT data and devices, transportation, medical devices and healthcare and upcoming 5G mobile networks to industrial applications in areas like manufacturing, automation, logistics, oil & gas and smart energy. Moreover, AI is embodied in different systems such as robots, drones, autonomous guided vehicles, smart wearables, intelligent cyber-physical systems and within a wide range of software-based systems such as chatbots.

The surge of interest in AI is largely due to recent advances in computing and storage. While the main principles of building AI systems have been around for over two decades, the technological advances facilitate the development of AI systems, as they enable the management of large datasets and speed up the execution of complex computations, especially across the cloud.



In this context, it's nowadays easier to build advanced deep learning systems that feature human-like reasoning, such as Google's AI engine that has repeatedly beaten human grandmasters in the Go game. At the same time, advances in smart sensors and cyber-physical systems facilitate real world data collection, and the embodiment of AI agents in smart objects.

What is knowledge representation?

Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. **But how machines do all these things comes under knowledge representation and reasoning**. Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.
- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent:

Following are the kind of knowledge which needs to be represented in AI systems:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

Types of knowledge

Following are the various types of knowledge:



1. Declarative Knowledge:

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarativesentences.
- It is simpler than procedural language.

2. Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

Check your Progress

Note a:Write your answers in the space given below

b. Compare your notes with those given at the end of the unit

3. Explain Inference Engine

………………………………………………………………………………………

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

4. Heuristic knowledge:

- Heuristic knowledge is representing knowledge of some experts in a filed or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

5. Structural knowledge:

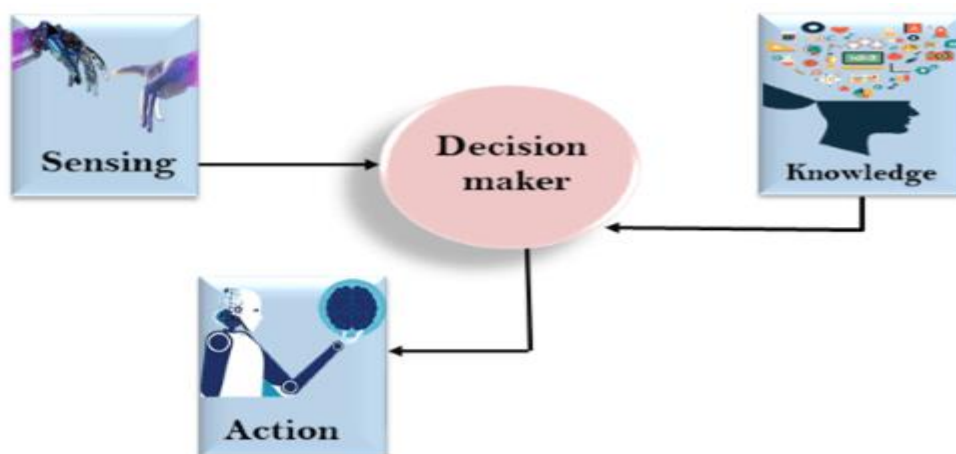
- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

The relation between knowledge and intelligence:

Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behavior in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

Let's suppose if you met some person who is speaking in a language which you don't know, then how you will able to act on that. The same thing applies to the intelligent behavior of the agents.

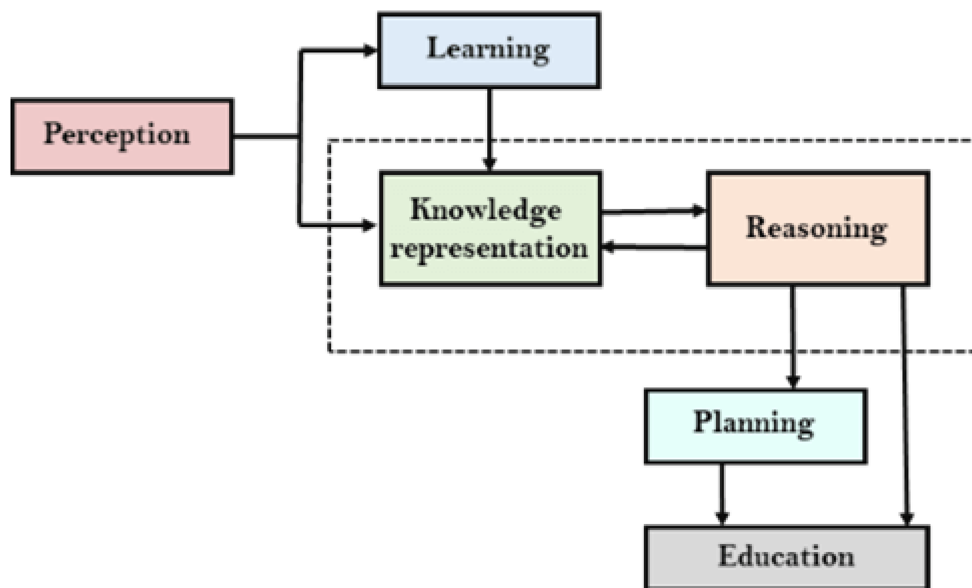
As we can see in below diagram, there is one decision maker which act by sensing the environment and using knowledge. But if the knowledge part will not present then, it cannot display intelligent behavior.



AI knowledge cycle:

An Artificial intelligence system has the following components for displaying intelligent behavior:

- Perception
- Learning
- Knowledge Representation and Reasoning
- Planning
- Execution



The above diagram is showing how an AI system can interact with the real world and what components help it to show intelligence. AI system has Perception component by which it retrieves information from its environment. It can be visual, audio or another form of sensory input. The learning component is responsible for learning from data captured by Perception comportment. In the complete cycle, the main components are knowledge representation and Reasoning. These two components are involved in showing the intelligence in machine-like humans. These two components are independent with each other but also coupled together. The planning and execution depend on analysis of Knowledge representation and reasoning.

Approaches to knowledge representation:

There are mainly four approaches to knowledge representation, which are givenbelow:

1. Simple relational knowledge:

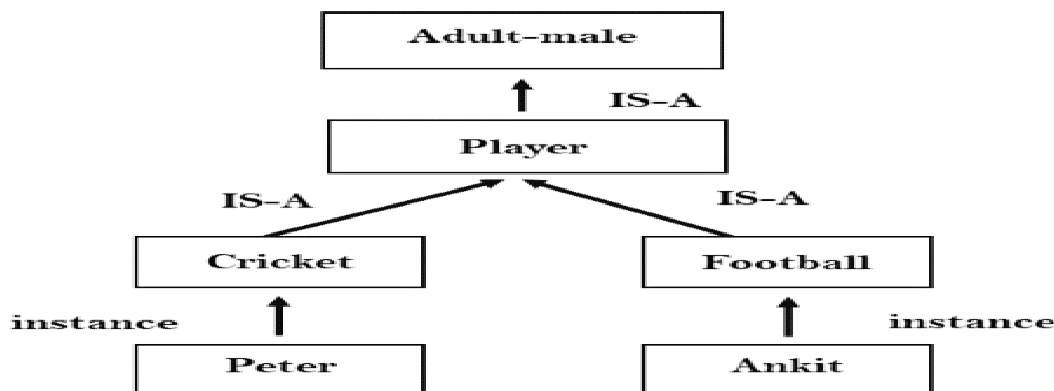
- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.

Example: The following is the simple relational knowledge representation.

Player	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

2. Inheritable knowledge:

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierarchal manner.
- In this approach, we apply inheritance property.
- Elements inherit values from other members of a class.
- This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- Every individual frame can represent the collection of attributes and its value.
- **Example:**



3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.
- **Example:** Let's suppose there are two statements:
    1. Marcus is a man
    2. All men are mortal
    Then it can represent as;

    man(Marcus)
    ∀x = man (x) ----------> mortal (x)s

4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

Requirements for knowledge Representation system:

A good knowledge representation system must possess the following properties.

1. **1. Representational Accuracy:**
   KR system should have the ability to represent all kind of required knowledge.
2. **2. Inferential Adequacy:**
   KR system should have ability to manipulate the representational structures to produce new knowledge corresponding to existing structure.
3. **3. Inferential Efficiency:**
   The ability to direct the inferential knowledge mechanism into the most productive directions by storing appropriate guides.
4. **4. Acquisitional efficiency-** The ability to acquire the new knowledge easily using automatic methods.

8.8 Unit – End Exercise

1. Explain Rule based systems
2 Describe the explanation module
3 Explain Inference Engine

8.9 Answers to Check Your Progress

1 Rule-based system is a system that is used to collect, store and manipulate knowledge to infer information. These systems are also known as production systems since it produces information from the knowledge gained.

2 This module is responsible for explaining the reasoning of the expert system to the user. When the user requests the reasoning of the system, this module will take in-charge in explaining it.

a) ***How query:*** The explanation module gives answer to this type of queries by using the sequence of rules triggered during a consolidation with the user. It clarifies how a rule is used and how a fact is inferred by the system.

b) ***Why query***: The system gives explanation why certain information is necessarily needed by the inference engine to complete a step-in reasoning method. For this type of query, the system acts in backward chaining. This type of explanation is dynamic. If the user wants to know the reason behind the query, the system will give explanations why the query was asked.

3. The inference engine replies to the user queries through I/O interface. It uses both the static knowledge and dynamic information. The inference engine module will get information from the knowledge base and applies it to find an answer to the problem. This inference engine makes inferring by analyzing which facts satisfies the rules. Then the satisfied rules will execute based on their priority. There are three stages in which the inferring process is carried out. They are match, select and execute.

8.10 Suggested Readings

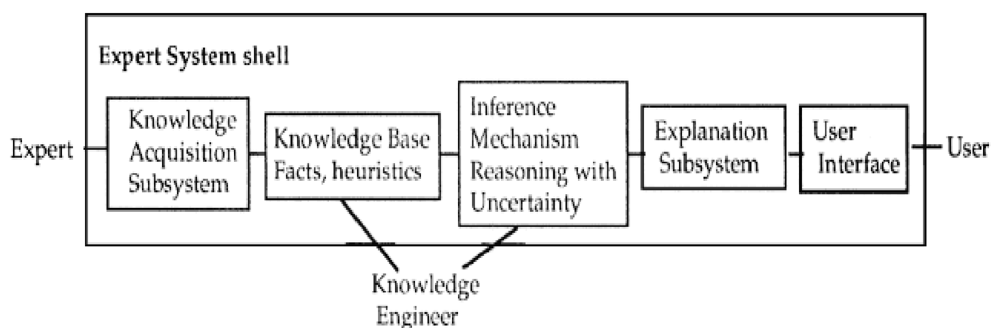
1. Ligęza, A.: Logical Foundations for Rule-based Systems, 2nd edn. Springer, Heidelberg (2006).
2. http://www.billbreitmayer.com/rule_based_systems/rule_based_design.html (accessed on February 10, 2011)
3. https://www.javatpoint.com/knowledge-representation-in-ai
4. https://cognitiveworld.com/articles/building-ai-works-domain-knowledge-and-impact-team-building
5. https://www.ques10.com/p/30481/representing-and-using-domain-knowledge/
6. https://www.brainkart.com/article/Knowledge-Base-(Representing-and-Using-Domain-Knowledge)_8597/

UNIT – IX Expert System Shell

Structure

9.1 Expert System Shell

An expert system shell is a software development platform containing the common components of expert systems. This shell consists of a prescribed method for developing applications by configuring and instantiating the basic expert system components. The expert system shell consists of the fundamental components such as knowledge base and reasoning engine. In addition, it consists of some generic components as shown in the figure below.



Expert System Shells

An Expert system shell is a software development environment. It contains the basic components of expert systems. A shell is associated with a prescribed method for building applications by configuring and instantiating these components.

Shell components and description

The generic components of a shell : the knowledge acquisition, the knowledge Base, the reasoning, the explanation and the user interface are shown below. The knowledge base and reasoning engine are the core components.

are computer programs that are derived from a branch of computer science research called *Artificial Intelligence* (AI). AI's scientific goal is to understand intelligence by building computer programs that exhibit intelligent behavior. It is concerned with the concepts and methods of symbolic inference, or reasoning, by a computer, and how the knowledge used to make those inferences will be represented inside the machine.

Of course, the term *intelligence* covers many cognitive skills, including the ability to solve problems, learn, and understand language; AI addresses all of those. But most progress to date in AI has been made in the area of problem solving -- concepts and methods for building programs that *reason* about problems rather than calculate a solution.

AI programs that achieve expert-level competence in solving problems in task areas by bringing to bear a body of knowledge about specific tasks are called *knowledge-based* or *expert systems*. Often, the term expert systems is reserved for programs whose knowledge base contains the knowledge used by human experts, in contrast to knowledge gathered from textbooks or non-experts. More often than not, the two terms, expert systems (ES) and knowledge-based systems (KBS), are used synonymously.

. Taken together, they represent the most widespread type of AI application. The area of human intellectual endeavor to be captured in an expert system is called the *task domain*. *Task* refers to some goal-oriented, problem-solving activity. *Domain* refers to the area within which the task is being performed. Typical tasks are diagnosis, planning, scheduling, configuration and design. An example of a task domain is aircraft crew scheduling, discussed in Chapter 2.

Building an expert system is known as *knowledge engineering* and its practitioners are called *knowledge engineers*. The knowledge engineer must make sure that the computer has all the knowledge needed to solve a problem. The knowledge engineer must choose one or more forms in which to represent the required knowledge as symbol patterns in the memory of the computer -- that is, he (or she) must choose a *knowledge representation*. He must also ensure that the computer can use the knowledge efficiently by selecting from a handful of *reasoning methods*. The practice of knowledge engineering is described later.

The Building Blocks of Expert Systems

Every expert system consists of two principal parts: the knowledge base; and the reasoning, or inference, engine.

The *knowledge base* of expert systems contains both factual and heuristic knowledge. *Factual knowledge* is that knowledge of the task domain that is widely shared, typically found in textbooks or journals, and commonly agreed upon by those knowledgeable in the particular field.

Heuristic knowledge is the less rigorous, more experiential, more judgmental knowledge of performance. In contrast to factual knowledge, heuristic knowledge is rarely discussed, and is largely individualistic. It is the knowledge of good practice, good judgment, and plausible reasoning in the field. It is the knowledge that underlies the "art of good guessing."

Knowledge representation formalizes and organizes the knowledge. One widely used representation is the *production rule*, or simply *rule*. A rule consists of an IF part and a THEN part (also called a *condition* and an *action*). The IF part lists a set of conditions in some logical combination. The piece of knowledge represented by the production rule is relevant to the line of reasoning being developed if the IF part of the rule is satisfied; consequently, the THEN part can be concluded, or its problem-solving action taken. Expert systems whose knowledge is represented in rule form are called *rule-based systems*.

9.2 Rules and Facts

A fact is a small piece of vital information. Facts alone are used only for limited purpose. The rules are very important to choose and apply facts to a user problem.

What is Prolog?

- Prolog stands for **Programming in logic.** It is used in artificial intelligence programming.
- Prolog is a **declarative** programming language. **For example:** While implementing the solution for a given problem, instead of specifying the ways to achieve a certain goal in a specific situation, user needs to specify about the situation (rules and facts) and the goal (query). After these stages, Prolog interpreter derives the solution.
- Prolog is useful in AI, NLP, databases but useless in other areas such as graphics or numerical algorithms.

Prolog facts

- A fact is something that seems to be true. **For example:** It's raining.
- In Prolog, facts are used to form the statements. Facts consist of a specific item or relation between two or more items.

How to convert English to prolog facts using facts and rules?

It is very simple to convert English sentence into Prolog facts. Some examples are explained in the following table.

English Statements	**Prolog Facts**
Dog is barking	barking(dog)
Jaya likes food if it is delicious.	likes(Jaya, Food):-delicious(Food)

.

Arithmetic Operations in Prolog

Prolog provides the facility for arithmetic operations.

As per the requirement of the user, arithmetic operations can be divided into some special purpose integer predicates and a series of general predicates for integer, floating point and rational arithmetic.

The general arithmetic predicates are handled by the expressions.

An expression is either a function or a simple number.

Prolog arithmetic is slightly different than other programming languages.
For example:

```
?- X is  2 + 1.
X = 3 ?
yes
```

In the above example, 'is' is used as a special predefined operator.

Operator precedence

If there is more than one operator in the arithmetic expression such as A-B*C+D, then the prolog decides an order in which the operator should be applied.

Prolog gives numerical value for each operator, operators with high precedence like '*' and '/' are applied before operators with relatively low precedence values like '+' and '-'.

Operator with same precedence value ('*' or '/') and ('+' or '-') should be applied from left to right.

So, the expression A-B*C+D can be written as A-(B*C)+D

Matching and Unification in Prolog

Definition: The two terms are said to be matched, if they are equal or if they consist of variables representing the resulting equal terms.
Prolog matches expressions in structural way. So,

```
?- 3 + 2= 5
no
```

Consider the following example,?- X + 3 = 2 * Y

But the following expressions will match because they have same structure.

Expression 1:
?- X + Y = 2 + 3
X = 2
Y = 3

Expression 2:
?- 2 + Y = X + 3
X = 2
Y = 3

Prolog Lists:

Lists are the finite sequence of elements.

Prolog uses [...] to build a list.

The notation [X|Y] represents that the first element is X and second element is Y (X is head and Y is tail).

Prolog has some special notation for lists:
I) [a]   [honda, maruti, renault]
ii) [a,b,c]   [pen, pencil, notebook]
iii) [] represents the empty list.

Example 1: Pattern Matching in Lists
?- [a,b] = [a,X]
X = b
but:
?- [a,b] = [X]
no

9.3 Knowledge Base

It is store where heuristic and factual knowledge are accumulated. The knowledge representation schemas are provided by the expert system tools. These schemas help in expressing knowledge about specific application domain. These expert system tools may consist of IF-THEN rules and Frames. For example, in PROLOG the knowledge is signified as logical statements.

Knowledge engineering

is the art of designing and building expert systems, and knowledge engineers are its practitioners. Gerald M. Weinberg said of programming in *The Psychology of Programming*: "'Programming,' -- like 'loving,' -- is a single word that encompasses an infinitude of activities" (Weinberg 1971). Knowledge engineering is the same, perhaps more so. We stated earlier that knowledge engineering is an applied part of the science of artificial intelligence which, in turn, is a part of computer science.

Today there are two ways to build an expert system. They can be built from scratch, or built using a piece of development software known as a "tool" or a "shell." Before we discuss these tools, let's briefly discuss what knowledge engineers do. Though different styles and methods of knowledge engineering exist, the basic approach is the same: a knowledge engineer interviews and observes a human expert or a group of experts and learns what the experts know, and how they reason with their knowledge. The engineer then translates the knowledge into a computer-usable language, and designs an inference engine, a reasoning structure, that uses the knowledge appropriately. He also determines how to integrate the use of uncertain knowledge in the reasoning process, and what kinds of explanation would be useful to the end user.

Next, the inference engine and facilities for representing knowledge and for explaining are programmed, and the domain knowledge is entered into the program piece by piece. It may be that the inference engine is not just right; the form of knowledge representation is awkward for the kind of knowledge needed for the task; and the expert might decide the pieces of knowledge are wrong. All these are discovered and modified as the expert system gradually gains competence.

The discovery and cumulation of techniques of machine reasoning and knowledge representation is generally the work of artificial intelligence research. The discovery and cumulation of knowledge of a task domain is the province of domain experts. Domain knowledge consists of both formal, textbook knowledge, and experiential knowledge -- the *expertise* of the experts.

Tools, Shells, and Skeletons

Compared to the wide variation in domain knowledge, only a small number of AI methods are known that are useful in expert systems. That is, currently there are only a handful of ways in which to represent knowledge, or to make inferences, or to generate explanations. Thus, systems can be built that contain these useful methods without any domain-specific knowledge. Such systems are known as *skeletal systems, shells,* or simply *AI tools*.

Building expert systems by using shells offers significant advantages. A system can be built to perform a unique task by entering into a shell all the necessary knowledge about a task domain. The inference engine that applies the knowledge to the task at hand is built into the shell. If the program is not very complicated and if an expert has had some training in the use of a shell, the expert can enter the knowledge himself.

Many commercial shells are available today, ranging in size from shells on PCs, to shells on workstations, to shells on large mainframe computers. They range in price from hundreds to tens of thousands of dollars, and range in complexity from simple, forward-chained, rule-based systems requiring two days of training to those so complex that only highly trained knowledge engineers can use them to advantage. They range from general-purpose shells to shells custom-tailored to a class of tasks, such as financial planning or real-time process control.

Although shells simplify programming, in general they don't help with knowledge acquisition. *Knowledge acquisition* refers to the task of endowing expert systems with knowledge, a task currently performed by knowledge engineers. The choice of reasoning method, or a shell, is important, but it isn't as important as the accumulation of high-quality knowledge. The power of an expert system lies in its store of knowledge about the task domain -- the more knowledge a system is given, the more competent it becomes.

Bricks and Mortar

The fundamental working hypothesis of AI is that intelligent behavior can be precisely described as symbol manipulation and can be modeled with the symbol processing capabilities of the computer.

In the late 1950s, special programming languages were invented that facilitate symbol manipulation. The most prominent is called LISP (LISt Processing). Because of its simple elegance and flexibility, most AI research programs are written in LISP, but commercial applications have moved away from LISP.

In the early 1970s another AI programming language was invented in France. It is called PROLOG (PROgramming in LOGic). LISP has its roots in one area of mathematics (lambda calculus), PROLOG in another (first-order predicate calculus).

PROLOG consists of English-like statements which are facts (assertions), rules (of inference), and questions. Here is an inference rule: "If object-x is part-of object-y then a component-of object-y is object-x."

Programs written in PROLOG have behavior similar to rule-based systems written in LISP. PROLOG, however, did not immediately become a language of choice for AI programmers. In the early 1980s it was given impetus with the announcement by the Japanese that they would use a logic programming language for the Fifth Generation Computing Systems (FGCS) Project. A variety of logic-based programming languages have since arisen, and the term *prolog* has become generic.

Check your Progress

Note: a. Write your answer in the space given below

b. Compare your answer with those given at the end of the unit.

i) Define Rules and Facts

………………………………………………………………………………………

………………………………………………………………………………………

9.4 Reasoning Engine

It consists of the inference mechanism or logic for manipulating the knowledge and the logical information in the knowledge base to produce a reasoning technique in solving a problem. This mechanism can range from simple modus ponens backward chaining of IF-THEN rules to case-based reasoning.

9.4.1 Logic: Logic is used in different forms to reason about the correctness of computational representation. The types of logic are,

- Programming language such as PROLOG (Programming in Logic)
- Calculus or propositional logic (consists of elementary sentences joined by ‘and’, ‘or’ and ‘not’)
- Predicate logic (objects and relations such as “is-a” and “has-a”)

9.5 Knowledge Acquisition Subsystem

The knowledge acquisition subsystem is handled by an expert to provide knowledge to the knowledge base component. In this case the expert will be a knowledge engineer who is capable of serving knowledge. Since this process is time consuming and acquires a lot of skill from the expert, it mostly limits the functioning and designing of the expert system in commercial environment.

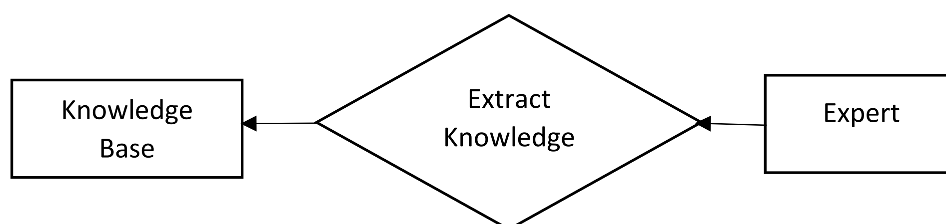


Figure Knowledge Acquisition Process

Information presented in this module is largely summarized from: Jones, P.H. 1989. Knowledge Acquisition. In: Barrett, J.R. and D.D. Jones. Knowledge Engineering in Agriculture. ASAE Monograph No. 8, ASAE, St. Joseph, MI.

Introduction

Knowledge acquisition is the process of extracting, structuring and organizing knowledge from one source, usually human experts, so it can be used in software such as an ES. This is often the major obstacle in building an ES.

There are three main topic areas central to knowledge acquisition that require consideration in all ES projects. First, the domain must be evaluated to determine if the type of knowledge in the domain is suitable for an ES. Second, the source of expertise must be identified and evaluated to ensure that the specific level of knowledge required by the project is provided. Third, if the major source of expertise is a person, the specific knowledge acquisition techniques and participants need to be identified.

Theoretical Considerations

An ES attempts to replicate in software the reasoning/pattern-recognition abilities of human experts who are distinctive because of their particular knowledge and specialized intelligence. ES should be heuristic and readily distinguishable from algorithmic programs and databases. Further, ES should be based on expert knowledge, not just competent or skillful behavior.

Domains

Several domain features are frequently listed for consideration in determining whether an ES is appropriate for a particular problem domain. Several of these caveats relate directly to knowledge acquisition. First, bona fide experts, people with generally acknowledge expertise in the domain, must exist. Second, there must be general consensus among experts about the accuracy of solutions in a domain. Third, experts in the domain must be able to communicate the details of their problem solving methods. Fourth, the domain should be narrow and well defined and solutions within the domain must not require common sense

Experts

Although an ES knowledge base can be developed from a range of sources such as textbooks, manuals and simulation models, the knowledge at the core of a well developed ES comes from human experts. Although multiple experts can be used, the ideal ES should be based on the knowledge of a single expert. In light of the pivotal role of the expert, caveats for choosing a domain expert are not surprising. First, the expert should agree with the goals of the project. Second, the expert should be cooperative and easy to work with. Third, good verbal communication skills are needed. Fourth, the expert must be willing and able to make the required time commitment (there must also be adequate administrative/managerial support for this too).

Knowledge Acquisition Technique

At the heart of the process is the interview. The heuristic model of the domain is usually extracted through a series of intense, systematic interviews, usually extending over a period of many months. Note that this assumes the expert and the knowledge engineer are not the same person. It is generally best that the expert and the knowledge engineer not be the same person since the deeper the experts' knowledge, the less able they are in describing their logic. Furthermore, in their efforts to describe their procedures, experts tend to rationalize their knowledge and this can be misleading.

General suggestions about the knowledge acquisition process are summarized in rough chronological order below:

1. Observe the person solving real problems.
2. Through discussions, identify the kinds of data, knowledge and procedures required to solve different types of problems.
3. Build scenarios with the expert that can be associated with different problem types.
4. Have the expert solve a series of problems verbally and ask the rationale behind each step.
5. Develop rules based on the interviews and solve the problems with them.
6. Have the expert review the rules and the general problem solving procedure.
7. Compare the responses of outside experts to a set of scenarios obtained from the project's expert and the ES.

Note that most of these procedures require a close working relationship between the knowledge engineer and the expert.

Practical Considerations

The preceding section provided an idealized version of how ES projects might be conducted. In most instances, the above suggestions are considered and modified to suit the particular project. The remainder of this section will describe a range of knowledge acquisition techniques that have been successfully used in the development of ES.

Operational Goals

After an evaluation of the problem domain shows that an ES solution is appropriate and feasible, then realistic goals for the project can be formulated. An ES's operational goals should define exactly what level of expertise its final product should be able to deliver, who the expected user is and how the product is to be delivered. If participants do not have a shared concept of the project's operational goals, knowledge acquisition is hampered.

Pre-training

Pre-training the knowledge engineer about the domain can be important. In the past, knowledge engineers have often been unfamiliar with the domain. As a result, the development process was greatly hindered. If a knowledge engineer has limited knowledge of the problem domain, then pre-training in the domain is very important and can significantly boost the early development of the ES.

Knowledge Document

Once development begins on the knowledge base, the process should be well documented. In addition to tutorial a document, a knowledge document that succinctly state the project's current knowledge base should be kept. Conventions should be established for the document such as keeping the rules in quasi-English format, using standard domain jargon, giving descriptive names to the rules and including supplementary, explanatory clauses with each rule. The rules should be grouped into natural subdivisions and the entire document should be kept current.

Scenarios

An early goal of knowledge acquisition should be the development of a series of well developed scenarios that fully describe the kinds of procedures that the expert goes through in arriving at different solutions. If reasonably complete case studies do not exist, then one goal of pre-training should be to become so familiar with the domain that the interviewer can compose realistic

scenarios. Anecdotal stories that can be developed into scenarios are especially useful because they are often examples of unusual interactions at the edges of the domain. Familiarity with several realistic scenarios can be essential to understanding the expert in early interviews and the key to structuring later interviews. Finally, they are ultimately necessary for validation of the system.

Interviews

Experts are usually busy people and interviews held in the expert's work environment are likely to be interrupted. To maximize access to the expert and minimize interruptions it can be helpful to hold meetings away from the expert's workplace. Another possibility is to hold meetings after work hours and on weekends. At least initially, audiotape recordings ought to be made of the interviews because often times notes taken during an interview can be incomplete or suggest inconsistencies that can be clarified by listening to the tape. The knowledge engineer should also be alert to fatigue and limit interviews accordingly.

In early interviews, the format should be unstructured in the sense that discussion can take its own course. The knowledge engineer should resist the temptation to impose personal biases on what the expert is saying. During early discussions, experts are often asked to describe the tasks encountered in the domain and to go through example tasks explaining each step. An alternative or supplemental approach is simply to observe the expert on the job solving problems without interruption or to have the expert talk aloud during performance of a task with or without interruption. These procedures are variations of protocol analysis and are useful only with experts that primarily use verbal thought processes to solve domain problems.

For shorter term projects, initial interviews can be formalized to simplify rapid prototyping. One such technique is a structured interview in which the expert is asked to list the variables considered when making a decision. Next the expert is asked to list possible outcomes (solutions) from decision making. Finally, the expert is asked to connect variables to one another, solutions to one another and variables to solutions through rules.

A third technique is card sorting. In this procedure, the knowledge engineer prepares a stack of cards with typical solutions to problems in the domain. The expert is asked to sort the cards according to some characteristic important to finding solutions to the problem. After each sort, the expert is asked to identify the sorting variable. After each sort, the expert is asked to repeat the process based on another variable. Note that this technique is usually not as effective as the 2 previous.

In large projects, later interviews cannot be expected to be as productive as early interviews. Typically, later interviews should become increasingly structured and follow a cyclical pattern where bits of knowledge are elicited, documented and tested. During this phase of knowledge acquisition, the interviewer must begin methodically to uncover the more subtle aspects of the knowledge. Typically, this process is based on scenarios. By modifying the scenarios in different ways, the interviewer can probe the expert's sensitivity.

During interviews, it may be helpful to work at a whiteboard to flexibly record and order the exact phraseology of rules or other representations. It may also be helpful to establish recording conventions for use such as color coding different aspects of a rule and using flags to note and defer consideration of significant but peripheral details.

Structured interviews should direct the course of a meeting to accomplish specific goals defined in advance. For instance, once a prototypic knowledge base is developed, the expert can be asked to evaluate it line by line. Other less obvious structures can be imposed on interviews, such as asking the expert to perform a task with limited information or during a limited period of time. Even these structured interviews can deviate from the session's intended goals. Sometimes such deviations show subtleties in the expert's procedures and at other times the interview simply becomes sidetracked, requiring the knowledge engineer to redirect the session.

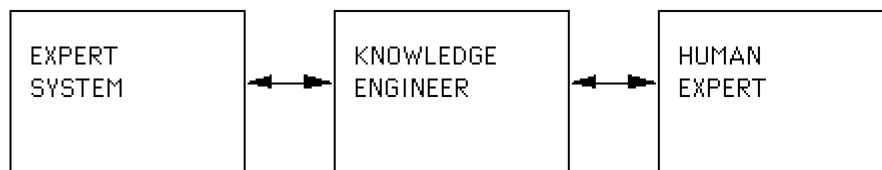
Questionnaires

When specific information is needed, a questionnaire can sometimes be used effectively. Questionnaires are generally used in combination with other techniques such as interviews.

An individual expert who is having ability to create, modify or add any changes to the knowledge base will use the knowledge base program. Some important sources such as textbooks and research reports are used as fundamentals of knowledge. A human expert and the user's own experience can also be used as potential basis of knowledge.

There are possibly two types of knowledge such as quantitative and qualitative on which the domain expert makes decisions. The knowledge has to be fed into the expert system in a suitable form. It is the responsibility of a system engineer to interpret the standard procedure into the appropriate form which the expert system can understand. This task is very difficult task where there is a chance of occurrence of bottleneck while construction expert system.

KNOWLEDGE ACQUISITION



The system must liaise with people in order to gain knowledge and the people must be specialised in the appropriate area of activity. For example medical doctors, geologists or chemists. The knowledge engineer acts as an intermediary between the specialist and the expert system. Typical of the information that must be gleaned is

vocabulary or jargon, general concepts and facts, problems that commonly arise, the solutions to the problems that occur and skills for solving particular problems. This process of picking the brain of an expert is a specialised form of data capture and makes use of interview techniques. The knowledge engineer is also responsible for the self consistency of the data loaded. Thus a number of specific tests have to be performed to ensure that the conclusions reached are sensible.

9.6 Explanation Subsystem

The explanation subsystem tries to explain the system's actions. These explanations can be used to clarify how the intermediate or final solutions were arrived at to justifying the need for extra data. This component answers how a certain conclusion is given to a problem. This component gives solution to the questions like:

- How the system takes a decision?
- On what basis does the system take the decision?
- Why the system addresses a question?

Explanation subsystem: A subsystem that explains the system's actions. The explanation can range from how the final or intermediate solutions were arrived at to justifying the need for additional data.

A subsystem that explains the system's actions. The explanation can range from how the final or intermediate solutions were arrived at justifying the need for additional data.

Explanation subsystem: This part of shell is responsible for explaining or justifying the final or intermediate result of user query. It is also responsible to justify need of additional knowledge.

Check your Progress
Note: a. Write your answer in the space given below
b. Compare your answer with those given at the end of the unit.
i. Explain IF-THEN rules.
………………………………………………………………………………………

………………………………………………………………………………………

9.7 User Interface

The user interface is the intermediate which helps to connect the user to the expert system. Through the user interface,

- The user can ask query to the system

- The user can give input to the system
- The user can get advice from the system

The expert system avails the communication between the user and the system. But still it is incapable to recognize the normal language and general knowledge. At times the expert system provides a user-friendly interaction with the user. Initially the expert system interface was only text based, and now specific expert systems are capable of providing Graphical User Interface (GUI).

A means of communication with the user. The user interface is generally not a part of the expert system technology. It was not given much attention in the past. However, the user interface can make a critical difference in the pe eived utility of an Expert system.

The E.S shell simplifies the process of creating a knowledge base. It is the shell that actually processes the information entered by a user relates it to the concepts contained in the knowledge base and provides an assessment or solution for a particular problem. Thus E.S shell provides a layer between the user interface and the computer O.S to manage the input and output of the data. It also manipulates the information provided by the user in conjunction with the knowledge base to arrive at a particular conclusion.

9.8 Unit – End Exercise

1. Define Rules and Facts.
2. Explain IF-THEN Logic.

9.9 Answers to Check Your Progress

1. Rules and Facts: - A fact is a small piece of vital information. Facts alone are used only for limited purpose. The rules are very important to choose and apply facts to a user problem.
2. Logic is used in different forms to reason about the correctness of computational representation. The types of logic are,
    - Programming language such as PROLOG (Programming in Logic)
    - Calculus or propositional logic (consists of elementary sentences joined by ‘and’, ‘or’ and ‘not’)
    - Predicate logic (objects and relations such as “is-a” and “has-a”)

9.10 Suggested Readings

1. Robert J. Mockler, Developing Knowledge-Based Systems Using an Expert System Shell, Macmillan Pub Co, January 1, 1992.
2. Chris Nikolopoulos, Expert Systems: Introduction to First and Second Generation and Hybrid Knowledge Based Systems, CRC Press, January 10, 1997.
3. https://www.tutorialride.com/artificial-intelligence/prolog-in-ai.htm
4. https://www.brainkart.com/article/Expert-System-Shells_8599/
5. https://www.brainkart.com/article/Expert-System-Shells_8599/
6. https://en.wikibooks.org/wiki/Category:Book:Expert_Systems
7. https://www.ques10.com/p/30483/write-a-note-on-expert-system-shell/
8. http://www.wtec.org/loyola/kb/c1_s1.htm
9. https://www.guru99.com/best-ai-chatbots.html

UNIT – X AI IN ROBOTICS

Structure

12.1 Introduction

Artificial Intelligence in Robotics

With the invention of machines or computers, their capability to perform different tasks went on increasing exponentially. Humans have developed the power of computer systems in terms of diverse working domains, with increasing speed, and reducing size with respect to time.

Robotics is a domain in artificial intelligence that deals with the study of creating intelligent and efficient robots. Robots are the artificial agents acting in real world environment.

Robots are aimed at manipulating the objects by perceiving, picking, moving, modifying the physical properties of object, destroying it, or to have an effect thereby freeing manpower from doing repetitive functions without getting bored, distracted, or exhausted.

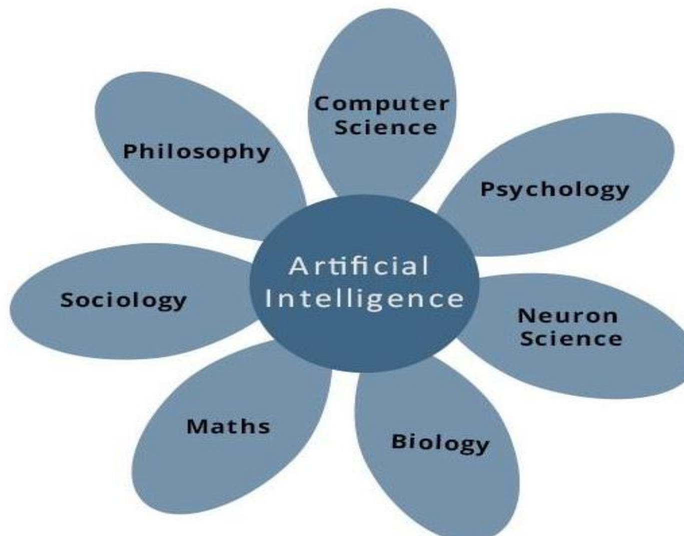
Goals of Artificial Intelligence

- **For Implementing Human Intelligence in Machines -** Creating systems that understand, learn, think and behave like humans.
- **For Developing Expert Systems -** The systems which exhibit intelligent behavior, learn, explain, demonstrate, and advice its users.

What Contributes to Artificial Intelligence

Artificial intelligence is a technology and science based on disciplines such as Psychology, Computer Science, Biology, Mathematics, Linguistics, and Engineering. A major thrust to artificial intelligence is the development of computer functions associated with human intelligence, such as learning, reasoning and problem solving.

- Consider the different areas which contribute to artificial intelligence are:-



Application of Artificial Intelligence (AI)

- **Expert Systems -** There are various applications which integrate machine, special information and software to impart advising and reasoning. These systems provide explanation and advice to the users.

- **Gaming -** AI plays major role in strategic games such as poker, chess, tic-tac-toe, etc. Using artificial intelligence the machine can think of large number of possible moves based on general knowledge.
- **Natural Language Processing -** Using natural language processing it is possible to interact with a computer that can understand natural language spoken by humans.
- **Vision systems -** These systems interpret, understand, and comprehend a visual input on the computer.
- **Intelligent Robots -** Robots are designed for performing the tasks given by a human. They have sensors embedded to detect physical data from the outside environment such as heat, light, sound, pressure, etc. They have multiple sensors, efficient processors and large memory, to exhibit intelligence. In addition, they are capable to learn from their mistakes and they can easily adapt to the new environment.

Difference in Robot System and Other AI Program

Here is the difference between the two −

AI Programs	Robots
They usually operate in computer-stimulated worlds.	They operate in real physical world
The input to an AI program is in symbols and rules.	Inputs to robots is analog signal in the form of speech waveform or images
They need general purpose computers to operate on.	They need special hardware with sensors and effectors.

Aspects of Robotics

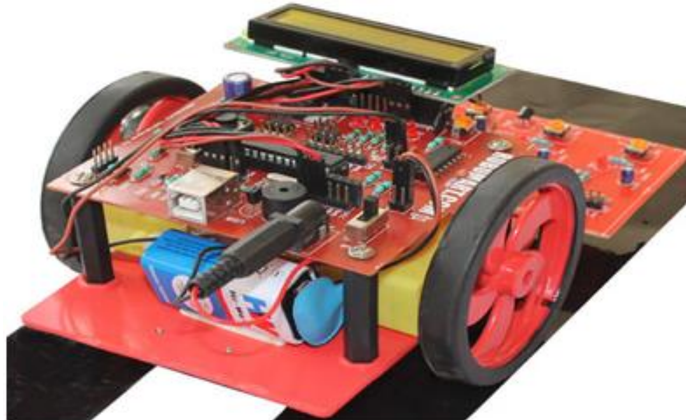
- The robots have **mechanical construction**, form, or shape designed to accomplish a particular task.
- They have **electrical components** which power and control the machinery.
- They contain some level of **computer program** that determines what, when and how a robot does something.

Types of Robots

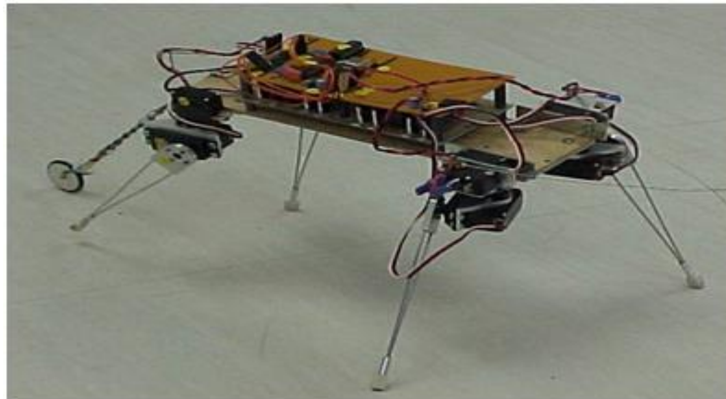
1**) Mobile Robots**

- Mobile robots are able to move from one location to another location using locomotion. It is an automatic machine that is capable of navigating an uncontrolled environment without any requirement of physical and electromechanical guidance devices. Mobile Robots are of two types:

(a) Rolling robots - Rolling robots require wheels to move around. They can easily and quickly search. But they are only useful in flat areas.



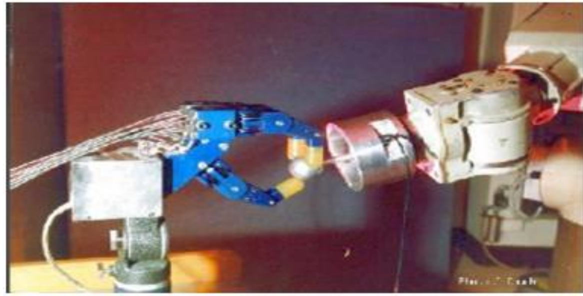
(b) Walking robots - Robots with legs are usually used in condition where the terrain is rocky. Most walking robots have at least 4 legs.



2) Industrial Robots

Industrial robots perform same tasks repeatedly without ever moving. These robots are working in industries in which there is requirement of performing dull and repeated tasks suitable for robot.

An industrial robot never tired, it will perform their works day and night without ever complaining.



3) Autonomous Robots

Autonomous robots are self-supported. They use a program that provides them the opportunity to decide the action to perform depending on their surroundings.

Using artificial intelligence these robots often learn new behavior. They start with a short routine and adapt this routine to be more successful in a task they perform. Hence, the most successful routine will be repeated.



4) Remote Controlled Robots

Remote controlled robot used for performing complicated and undetermined tasks that autonomous robot cannot perform due to uncertainty of operation.

Complicated tasks are best performed by human beings with real brainpower. Therefore a person can guide a robot by using remote. Using remote controlled operation human can perform dangerous tasks without being at the spot where the tasks are performed.

Let's see a NASA robot designed to explore volcanoes via remote control:



Check your Progress-1

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

1. Define Artificial intelligence in robotics

……………………………………………………………

2. Describe autonomous robots.

………………………………………………………….

10.2 StateSpaceSearch

State Space Search is a process used in the field of computer science, including artificial intelligence (AI), in which successive configurations or *states* of an instance are considered, with the intention of finding a *goal state* with a desired property.

Problems are often modelled as a state space, a set of *states* that a problem can be in. The set of states forms a graph where two states are connected if there is an *operation* that can be performed to transform the first state into the second.

In state space search a state space is formally represented as a tuple , in which:

- is the set of all possible states;
- is the set of possible action, not related to a particular state but regarding all the state space;
- is the function that establish which action is possible to perform in a certain state;
- is the function that return the state reached performing action in state
- Is the cost of performing an action in state, in many state spaces is a constant, but this is not true in general.

Search Algorithm Terminologies:

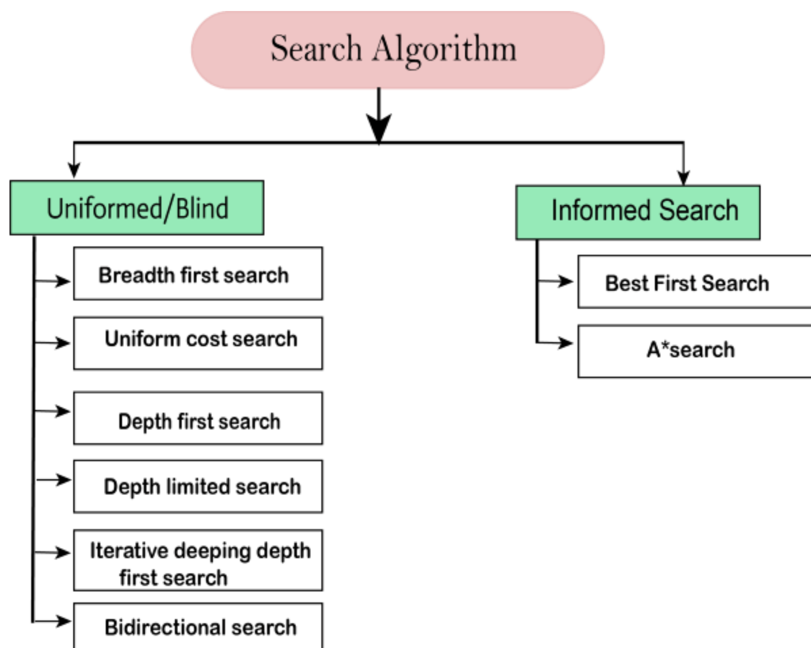
- **Search:** Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
  a. **Search Space:** Search space represents a set of possible solutions, which a system may have.

b. **Start State:** It is a state from where agent begins **the search**.
c. **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.

- **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
- **Actions:** It gives the description of all the available actions to the agent.
- **Transition model:** A description of what each action do, can be represented as a transition model.
- **Path Cost:** It is a function which assigns a numeric cost to each path.
- **Solution:** It is an action sequence which leads from the start node to the goal node.
- **Optimal Solution:** If a solution has the lowest cost among all solutions.

Types of search algorithms

Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms.



Uninformed/Blind Search:

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search. It examines each node of the tree until it achieves the goal node.

It can be divided into five main types:

- Breadth-first search
- Uniform cost search
- Depth-first search
- Iterative deepening depth-first search
- Bidirectional Search

Informed Search

Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a Heuristic search.

A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.

Informed search can solve much complex problem which could not be solved in another way.

An example of informed search algorithms is a traveling salesman problem.

1. Greedy Search
2. A* Search

10.2.1 Uninformed Search Algorithms

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

1. Breadth-first Search
2. Depth-first Search
3. Depth-limited Search
4. Iterative deepening depth-first search
5. Uniform cost search
6. Bidirectional Search

1. Breadth-first Search:

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.

Advantages:

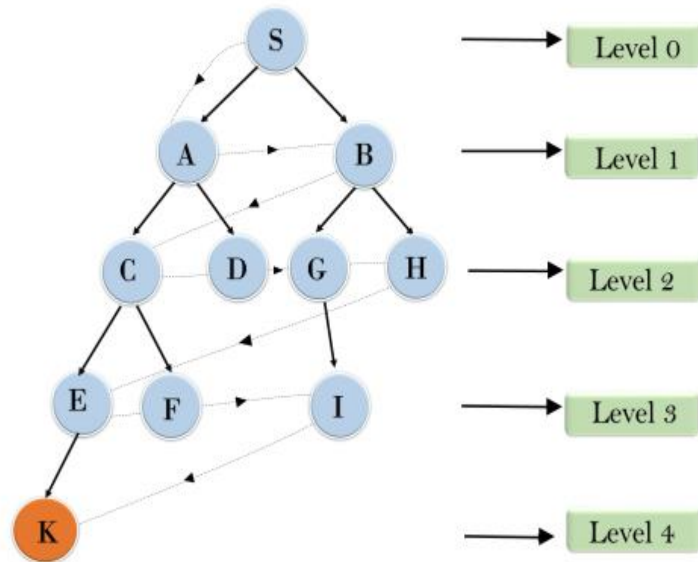
- BFS will provide a solution if any solution exists.
- If there are more than one solution for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Example:
In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

S---> A--->B---->C--->D---->G--->H--->E---->F---->I---->K



Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

$T(b) = 1+b^2+b^3+.......+ b^d= O(b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

2. Depth-first Search

- Depth-first search is a recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

Advantage:

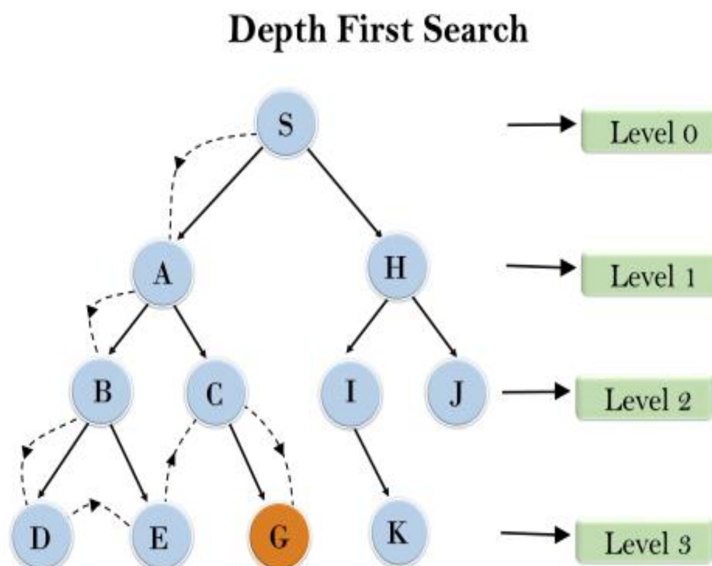
- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Example:

- In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:
- Root node--->Left node ----> right node.
- It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$T(n)= 1+ n^2+ n^3 +.........+ n^m=O(n^m)$

Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

3. Depth-Limited Search Algorithm:

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

- Standard failure value: It indicates that problem does not have any solution.
- Cutoff failure value: It defines no solution for the problem within a given depth limit.

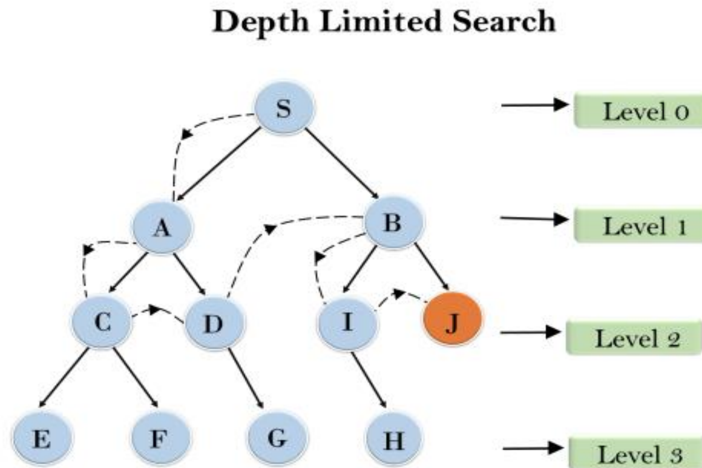
Advantages:

Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

Example:



Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is $\mathbf{O(b^{\ell})}$.

Space Complexity: Space complexity of DLS algorithm is O$\mathbf{(b \times \ell)}$.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if $\ell > d$.

4. Uniform-cost Search Algorithm:

Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost. Uniform-cost search expands nodes according to their path costs form the root node. It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

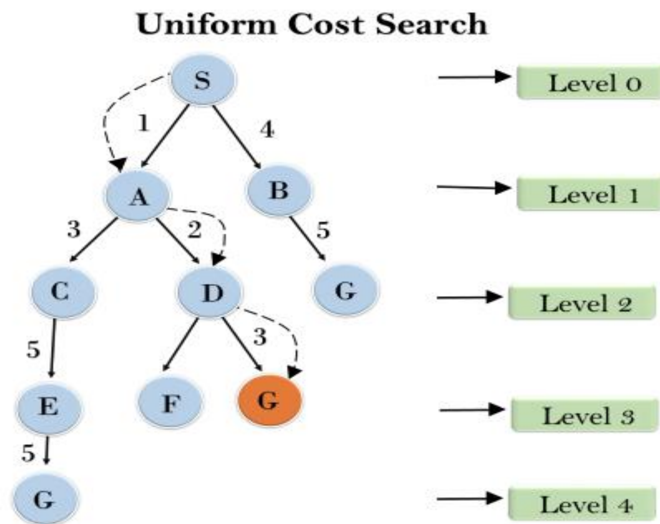
Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

Example:



Completeness:
Uniform-cost search is complete, such as if there is a solution, UCS will find it.

Time Complexity:

Let C* **is Cost of the optimal solution**, and **ε** is each step to get closer to the goal node. Then the number of steps is = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε.

Hence, the worst-case time complexity of Uniform-cost search is**O(b$^{1 + [C*/ε]}$)/**.

Space Complexity:

The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is **O(b$^{1 + [C*/ε]}$)**.

Optimal:

Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

5. Iterative deepeningdepth-first Search:

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

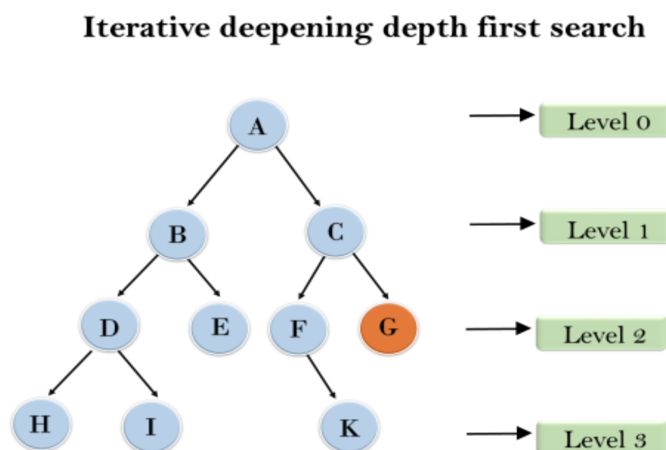
Advantages:

- Itcombines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example:
Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:



1'st Iteration-----> A
2'nd Iteration----> A, B, C
3'rd Iteration------>A, B, D, E, C, F, G
4'th Iteration------>A, B, D, H, I, E, C, F, K, G
In the fourth iteration, the algorithm will find the goal node.

Completeness:
This algorithm is complete is ifthe branching factor is finite.

Time Complexity:
Let's suppose b is the branching factor and depth is d then the worst-case time complexity is $\mathbf{O(b^d)}$.

Space Complexity:

The space complexity of IDDFS will be **O(bd)**.

Optimal:
IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

6. Bidirectional Search Algorithm:
Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex. The search stops when these two graphs intersect each other.

Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

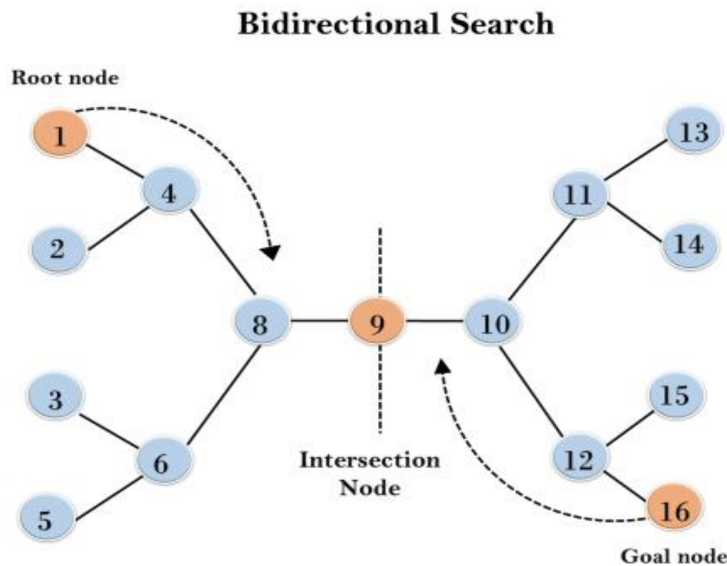
Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- **In bidirectional search, one should know the goal state in advance.**

Example:

In the below search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.

The algorithm terminates at node 9 where two searches meet.



Completeness: Bidirectional Search is complete if we use BFS in both searches.
Time Complexity: Time complexity of bidirectional search using BFS is $\mathbf{O(b^d)}$.
Space Complexity: Space complexity of bidirectional search is $\mathbf{O(b^d)}$.
Optimal: Bidirectional search is Optimal.

10.2.1 Informed Search Algorithms

So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space. But informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge helps agents to explore less to the search space and find more efficiently the goal node.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Heuristics function: Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time.

Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as:

h(n) <= h*(n)

Here h(n) is heuristic cost, and h*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

Pure Heuristic Search:

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found.

In the informed search we will discuss two main algorithms which are given below:

- **Best First Search Algorithm(Greedy search)**
- **A* Search Algorithm**

1) Best-first Search Algorithm (Greedy Search):

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

f(n)= g(n).

Were, h(n)= estimated cost from node n to the goal.

The greedy best first algorithm is implemented by the priority queue.

Best first search algorithm:

Step 1: Place the starting node into the OPEN list.

Step 2: If the OPEN list is empty, Stop and return failure.

Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.

Step 4: Expand the node n, and generate the successors of node n.

Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

Step 6: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.

Step 7: Return to Step 2.

Advantages:

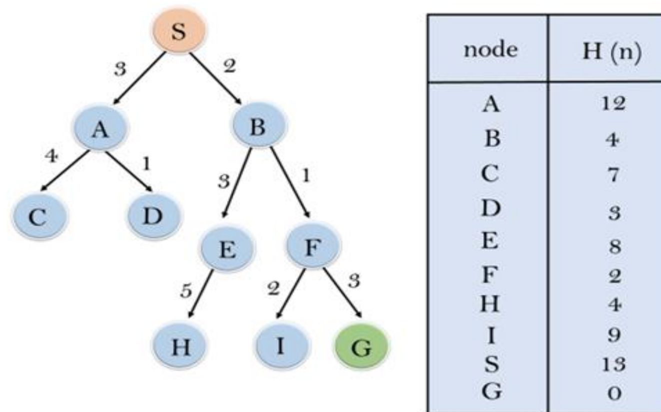
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

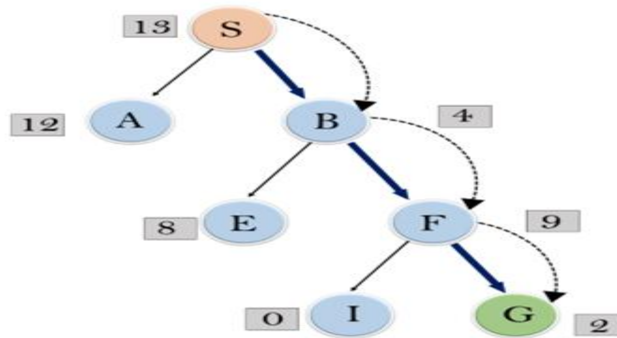
Example:

Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.



Expand the nodes of S and put in the CLOSED list

Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]

Iteration 2: Open [E, F, A], Closed [S, B]
: Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F]
: Open [I, E, A], Closed [S, B, F, G]

Hence the final solution path will be: **S----> B----->F----> G**

Time Complexity: The worst case time complexity of Greedy best first search is $O(b^m)$.

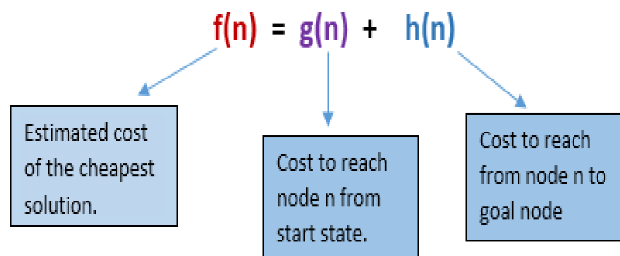
Space Complexity: The worst case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.

Complete: Greedy best-first search is also incomplete, even if the given state space is finite.
Optimal: Greedy best first search algorithm is not optimal.

2) A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

Advantages:

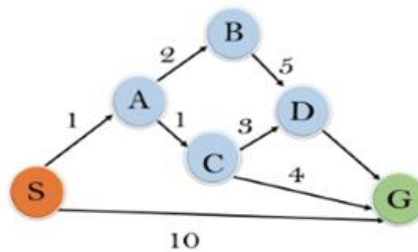
- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

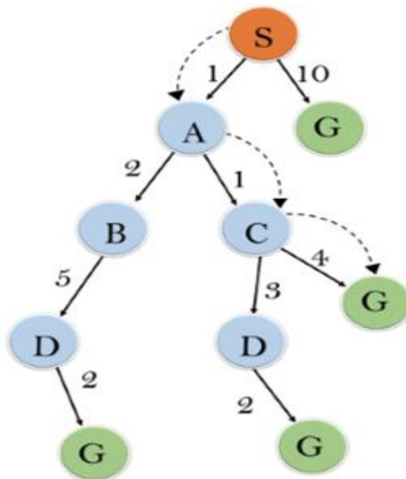
Example:

In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.
Here we will use OPEN and CLOSED list.



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution:



Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

Check your Progress-2

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

1. Define State Space Search

…………………………………………………………………

…………………………………………………………………

2. Describe best first search algorithm

…………………………………………………………………

…………………………………………………………………

10.3 Block world& robot example

What is the Blocks World? -- The world consists of:

- A flat surface such as a tabletop
- An adequate set of identical blocks which are identified by letters.
- The blocks can be stacked one on one to form towers of apparently unlimited height.
- The stacking is achieved using a robot arm which has fundamental operations and states which can be assessed using logic and combined using logical operations.
- The robot can hold one block at a time and only one block can be moved at a time.

We shall use the four actions:

UNSTACK(A,B)
-- pick up clear block A from block B;
STACK(A,B)
-- place block A using the arm onto clear block B;
PICKUP(A)
-- lift clear block A with the empty arm;
PUTDOWN(A)
-- place the held block A onto a free space on the table.

and the five predicates:

ON(A,B)
-- block A is on block B.
ONTABLE(A)
-- block A is on the table.
CLEAR(A)
-- block A has nothing on it.
HOLDING(A)
-- the arm holds block A.
ARMEMPTY
-- the arm holds nothing.

Using logic but not logical notation we can say that If the arm is holding a block it is not empty If block A is on the table it is not on any other block If block A is on block B,block B is not clear.

Why Use the Blocks world as an example?

The blocks world is chosen because:

- it is sufficiently simple and well behaved.
- easily understood
- yet still provides a good sample environment to study planning:
    - problems can be broken into nearly distinct subproblems
    - we can show how partial solutions need to be combined to form a realistic complete solution.

Planning

Planning refers to the process of computing several steps of a problem solving before executing any of them.Planning is useful as a problem solving technique for non decomposable problem.

Components of Planning System:

In any general problem solving systems, elementary techniques to perform following functions are required

- Choose the best rule (based on heuristics) to be applied
- Apply the chosen rule to get new problem state
- Detect when a solution has been found
- Detect dead ends so that new directions are explored.

To choose the rules,

- first isolate a set of differences between the desired goal state and current state,
- identify those rules that are relevant to reducing these difference,
- if more rules are found then apply heuristic information to choose out of them.

To apply rules,

In simple problem solving system,

- applying rules was easy as each rule specifies the problem state that would result from its application.
- In complex problem we deal with rules that specify only a small part of the complete problem state.

Let us consider the famous problem name as **Block World Problem ,**which helps to understand the importance of planning in artificial intelligent system.

The block world environment has ,

- Square blocks of same size
- Blocks can be stacked one upon another.
- Flat surface (table) on which blocks can be placed.
- Robot arm that can manipulate the blocks. It can hold only one block at a time.

In block world problem, the state is described by a set of predicates representing the facts that were true in that state.One must describe for every action, each of the changes it makes to the state description.In addition, some statements that everything else remains unchanged is also necessary.We are having four types of operations done by robot in block world environment .They are

UNSTACK (X, Y) : [US (X, Y)]

- Pick up X from its current position on block Y. The arm must be empty and X has no block on top of it.

STACK (X, Y): [S (X, Y)]

- Place block X on block Y. Arm must holding X and the top of Y is clear.

PICKUP (X): [PU (X)]

- Pick up X from the table and hold it. Initially the arm must be empty and top of X is clear.

PUTDOWN (X): [PD (X)]

- Put block X down on the table. The arm must have been holding block X.

Along with the operations ,some predicates to be used to describe an environment clearly.Those predicates are,

- ON(X, Y) - Block X on block Y.
- ONT() - Block X on the table.
- CL(X) - Top of X clear.
- HOLD(X) - Robot-Arm holding X.
- AE - Robot-arm empty.

Logical statements true in this block world.

X Holding X means, arm is not empty

($ X) HOLD (X) ® ~ AE

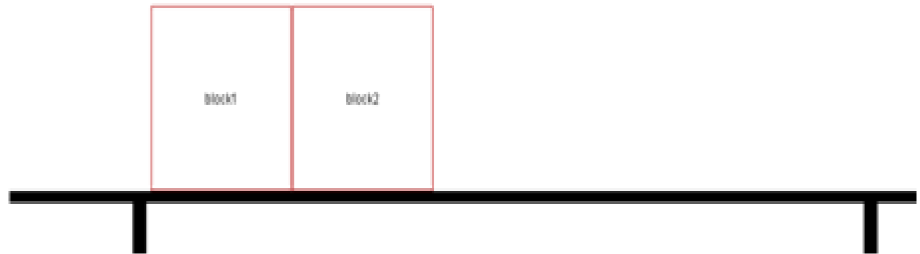
X is on a table means that X is not on the top of any block

(" X) ONT (X) ® ~ ($ Y) ON (X, Y)

Any block with no block on has clear top

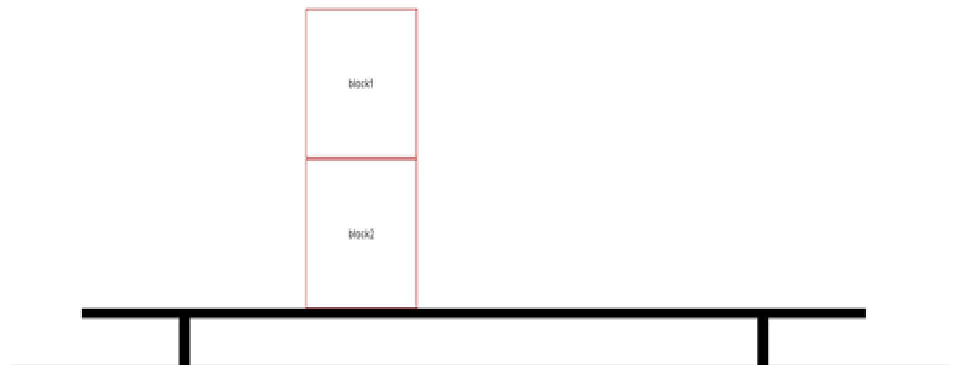
(" X) (~ ($ Y) ON (Y,X)) ® CL (X)

Initial State

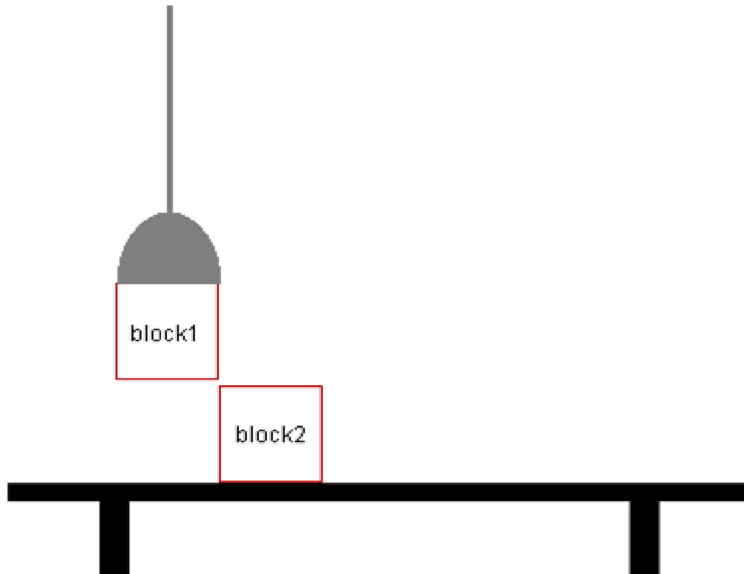


Armempty
clear(block2)
ontable(block2)
ontable(block1)
clear(block1)

Goal State



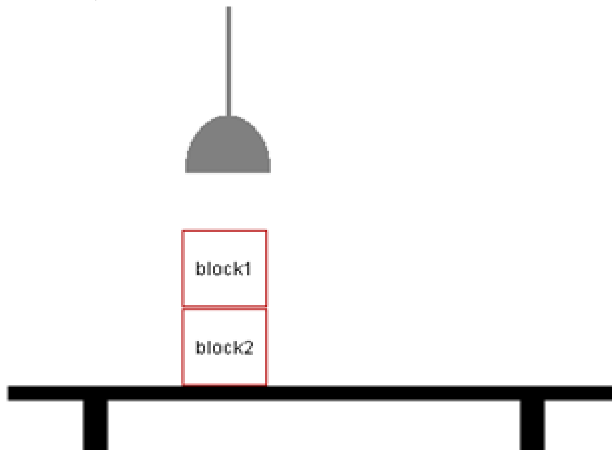
Armempty
ontable(block2)
on(block1, block2)
clear(block1)
We have to generate a plan to reach goal state from initial state given.In this example the initial state has two blocks Block1 and Block 2.Both is placed on table.To reach the goal state first we have to
PICKUP(Block 1)



We need to check whether we reach goal state or not ,after completion of each and every operation.Here the environment looks like,

Hold(block1)
Clear(Block2)
OnTable(Block2)

This is not the goal state .so ,we have to continue the process. Next the block 1 needs to be place on block 2,to achieve this do the operation **STACK(Block1,Block2).** After this operation the environment looks like,



ArmEmpty,on(Block1,Block2),Clear(Block1),OnTable(Block2)

We reach the goal state,the plan for reaching goal state is
PICKUP(Block1) and Stack(Block1,Block2)

Check your Progress-3

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

1. Define Block world

……………………………………………………………..

………………………………………………………………

2. Why Use the Blocks world as an example?

………………………………………………………………

……………………………………………………………..

10.4 Path Selection

Graph Searching and the Generic Search Algorithm

Many AI problems can be cast as the problem of finding a path in a graph. A graph is made up of nodes and arcs. Arcs are ordered pairs of nodes that can have associated costs.

Suppose we have a set of nodes that we call "start nodes" and a set of nodes that we call "goal nodes", a **solution** is a path from a start node to a goal node.

Consider the following simple graph (this is a tree as there is at most one arc going into each node). The start nodes are colored grey, the goal nodes as are colored yellow, and the other nodes are not coloured.

To find a solution, we need to search for a path. We use the generic searching algorithm. The **frontier** is a set of paths from a start node (we often identify the path with the node at the end of the path). The nodes at the end of the frontier are outlined in green or blue. Initially the frontier is the set of empty paths from start nodes. Intuitively the generic graph searching algorithm is:

- Repeat
  - select a path on the frontier. Let's call the path selected *P*.
  - if *P* is a path to a goal node, stop and return P,
  - remove *P* from the frontier
  - for each neighbor of the node at the end of *P*, extend *P* to that neighbour and add the extended path to the frontier
- Until the frontier is empty. When it is empty there are no more solutions.

To see how this works you can carry out the generic search algorithm selecting the nodes manually. The frontier is initially all coloured in green. You can click on a node on the frontier to select it. The node and the path to it turn red, and its neighbors (given in blue) are added to the frontier. The new frontier is then the nodes outlined in blue and green; the blue outlined nodes are the newly added nodes, and the green outlined nodes are the other node on the frontier. You can keep clicking on nodes till you find a solution. Then you can reset the search to try a different node ordering.

There are a number of features that should be noticed about this:

- For a finite graph without cycles, it will eventually find a solution no matter which order you select paths on the frontier.
- Some strategies for selecting paths from the frontier expand fewer nodes that other strategies.

Check your Progress-4

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

1. Define Path selection

………………………………………………………………..

…………………………………………………………………

2. What is meant by frontier?

…………………………………………………………………

10.5 Monkey & Banana Problem

Example:
There is a monkey at the door of a room. In the middle of the room a banana hangs In the middle of the room a banana hangs from the ceiling. The monkey wants it, but from the ceiling. The monkey wants it, but cannot jump high enough from the floor. At the window of the room there is a box that At the window of the room there is a box that the monkey can use.

The monkey can perform the following The monkey can perform the following actions: actions: ‹
Walk on the floor Walk on the floor ‹

Climb the box

Push the box around (if it is beside the box)

Grasp the banana if it is standing on the box directly under the banana.

We define the state as a 4 We define the state as a 4-tuple: (monkey (monkey-at, on-floor, box floor, box-at, has at, has-banana)

The program
The order of the rules is important (Why?) The order of the rules is important (Why?)

move(state(middle, onbox, middle, , middle, hasnot), grasp, state(middle, onbox, middle, has)).

move(state(P, onfloor onfloor, P, H), climb, state(P, , onbox, P, H)).

move(state(P1, onfloor onfloor, P1, H), , P1, H), push(P1, P2), state(P2, onfloor onfloor, P2, H)).

Move (state(P1, onfloor onfloor, B, H),

walk(P1, P2), state(P2, onfloor onfloor, B, H)).

canget(state(_, _, _, has)).

canget(State1) :-

move(State1, Move, State2),

canget(State2). (State2)

?- canget(state(atdoor, onfloor onfloor, atwindow atwindow, hasnot)).

10.6 AND OR graph

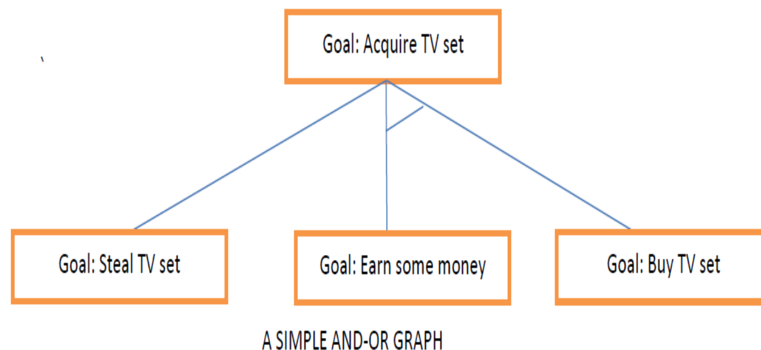
Problem Reduction:

So far we have considered search strategies for OR graphs through which we want to find a single path to a goal. Such structure represent the fact that we know how to get from anode to a goal state if we can discover how to get from that node to a goal state along any one of the branches leaving it.

And-Or Graphs

The AND-OR GRAPH (or tree) is useful for representing the solution of problems that can solved by decomposing them into a set of smaller problems, all of which must then be solved. This decomposition, or reduction, generates arcs that we call AND arcs. One AND arc may point to any number of successor nodes, all of which must be solved in order for the arc to point to a solution. Just as in an OR graph, several arcs may emerge from a single node, indicating a variety of ways in which the original problem might be solved.

Example For And-Or Graph



A SIMPLE AND-OR GRAPH

ALGORITHM:

1. Let G be a graph with only starting node INIT.
2. Repeat the followings until INIT is labeled SOLVED or h(INIT) > FUTILITY

Select an unexpanded node from the most promising path from INIT (call it NODE)

Generate successors of NODE. If there are none, set h(NODE) = FUTILITY (i.e., NODE is unsolvable); otherwise for each SUCCESSOR that is not an ancestor of NODE do the following:

i. Add SUCCESSSOR to G.

ii. If SUCCESSOR is a terminal node, label it SOLVED and set h(SUCCESSOR) = 0.

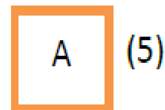
iii. If SUCCESSPR is not a terminal node, compute its h

Propagate the newly discovered information up the graph by doing the following: let S be set of SOLVED nodes or nodes whose h values have been changed and need to have values propagated back to their parents. Initialize S to Node. Until S is empty repeat the followings:

i. Remove a node from S and call it CURRENT.
ii. Compute the cost of each of the arcs emerging from CURRENT. Assign minimum cost of its successors as its h.
iii. Mark the best path out of CURRENT by marking the arc that had the minimum cost in step ii
iv. Mark CURRENT as SOLVED if all of the nodes connected to it through new labeled arc have been labeled SOLVED
v. If CURRENT has been labeled SOLVED or its cost was just changed, propagate its new cost back up through the graph. So add all of the ancestors of CURRENT to S.

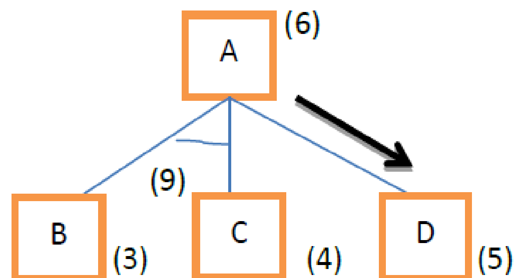
EXAMPLE: 1

STEP 1:



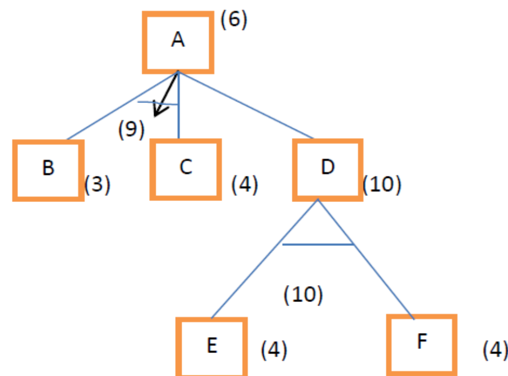
A is the only node, it is at the end of the current best path. It is expanded, yielding nodes B, C, D. The arc to D is labeled as the most promising one emerging from A, since it costs 6compared to B and C, Which costs 9.

STEP 2:



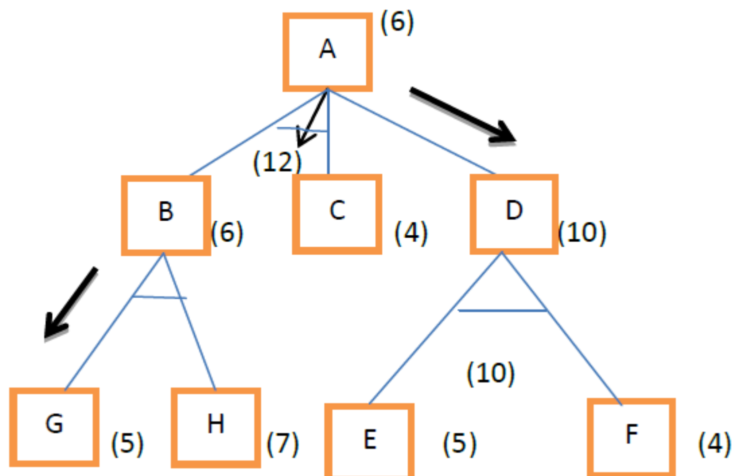
Node B is chosen for expansion. This process produces one new arc, the AND arc to E and F, with a combined cost estimate of 10.so we update the f' value of D to 10.Going back one more level, we see that this makes them AND arc B-C better than the arc to D, so it is labeled as the current best path.

STEP 3:

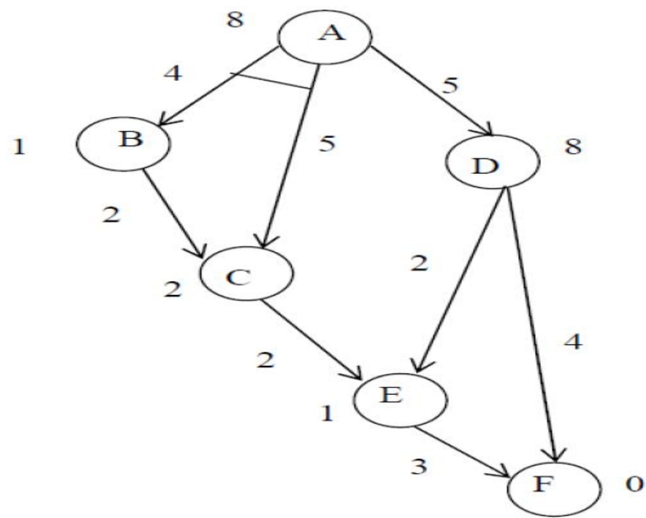


We traverse the arc from A and discover the unexpanded nodes B and C. If we going to find a solution along this path, we will have to expand both B and C eventually, so let's choose to explore B first. This generates two new arcs, the ones to G and to H. Propagating their f' values backward, we update f' of B to 6(since that is the best we think we can do, which we can achieve by going through G). This requires updating the cost of the AND arc B-C to 12(6+4+2). After doing that, the arc to D is again the better path from A, so we record that as the current best path and either node E or node F will chosen for expansion at step 4.

STEP 4:



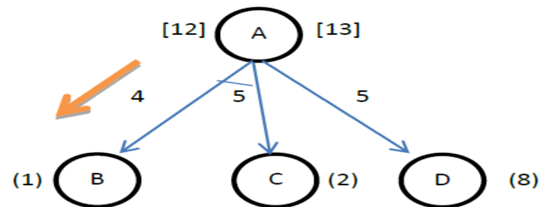
EXAMPLE: 2



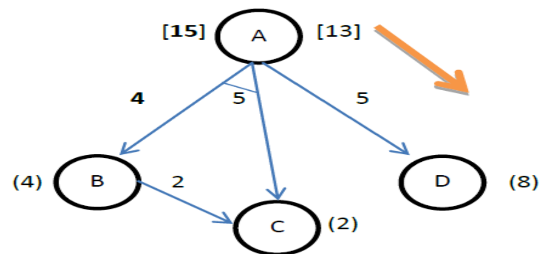
STEP1:



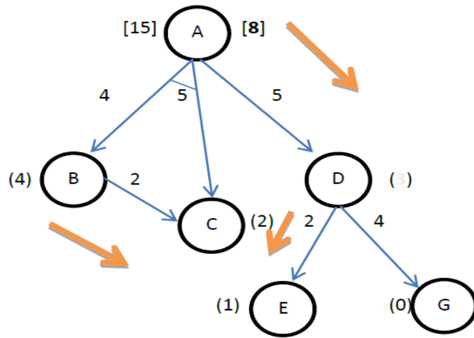
Step2:



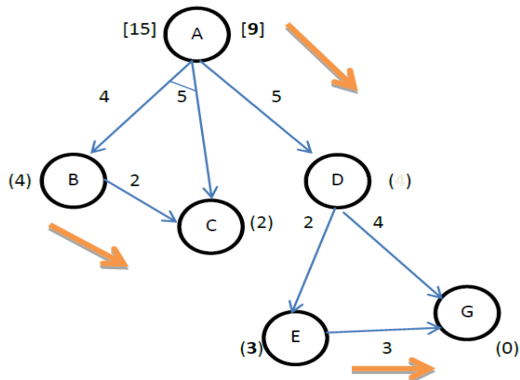
STEP3:



STEP4:



STEP4:



Check your Progress-5

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

1. Define AND OR Graph

……………………………………………………………………

……………………………………………………………………

2. Give the algorithm of AND OR Graph

……………………………………………………………………

10.7 MEANS - ENDS ANALYSIS

Means-Ends Analysis in Artificial Intelligence

- We have studied the strategies which can reason either in forward or backward, but a mixture of the two directions is appropriate for solving a complex and large problem. Such a mixed strategy, make it possible that first to solve the major part of a problem and then go back and solve the small problems arise during combining the big parts of the problem. Such a technique is called **Means-Ends Analysis**.
- Means-Ends Analysis is problem-solving techniques used in Artificial intelligence for limiting search in AI programs.
- It is a mixture of Backward and forward search technique.
- The MEA technique was first introduced in 1961 by Allen Newell, and Herbert A. Simon in their problem-solving computer program, which was named as General Problem Solver (GPS).
- The MEA analysis process centered on the evaluation of the difference between the current state and goal state.

How means-ends analysis Works:

The means-ends analysis process can be applied recursively for a problem. It is a strategy to control search in problem-solving. Following are the main Steps which describes the working of MEA technique for solving a problem.

a. First, evaluate the difference between Initial State and final State.
b. Select the various operators which can be applied for each difference.
c. Apply the operator at each difference, which reduces the difference between the current state and goal state.

Operator Subgoaling

In the MEA process, we detect the differences between the current state and goal state. Once these differences occur, then we can apply an operator to reduce the differences. But sometimes it is possible that an operator cannot be applied to the current state. So we create the subproblem of the current state, in which operator can be applied, such type of backward chaining in which operators are selected, and then sub goals are set up to establish the preconditions of the operator is called **Operator Subgoaling**.

Algorithm for Means-Ends Analysis:

Let's we take Current state as CURRENT and Goal State as GOAL, then following are the steps for the MEA algorithm.

Step 1: Compare CURRENT to GOAL, if there are no differences between both then return Success and Exit.

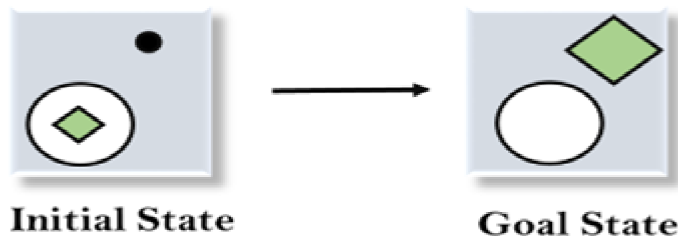
Step 2: Else, select the most significant difference and reduce it by doing the following steps until the success or failure occurs.

a. Select a new operator O which is applicable for the current difference, and if there is no such operator, then signal failure.
b. Attempt to apply operator O to CURRENT. Make a description of two states.
   i) O-Start, a state in which O?s preconditions are satisfied.
   ii) O-Result, the state that would result if O were applied In O-start.
c. If
   (First-Part <------ MEA (CURRENT, O-START)
   And
   (LAST-Part <----- MEA (O-Result, GOAL), are successful, then signal Success and return the result of combining FIRST-PART, O, and LAST-PART.

The above-discussed algorithm is more suitable for a simple problem and not adequate for solving complex problems.

Example of Mean-Ends Analysis:

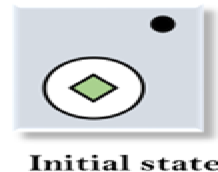
Let's take an example where we know the initial state and goal state as given below. In this problem, we need to get the goal state by finding differences between the initial state and goal state and applying operators.



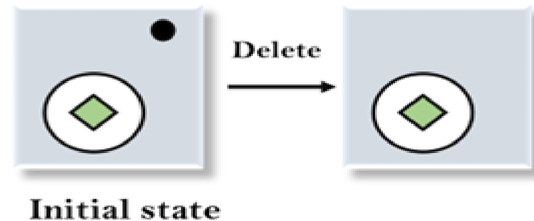
Solution:
To solve the above problem, we will first find the differences between initial states and goal states, and for each difference, we will generate a new state and will apply the operators. The operators we have for this problem are:

- **Move**
- **Delete**
- **Expand**

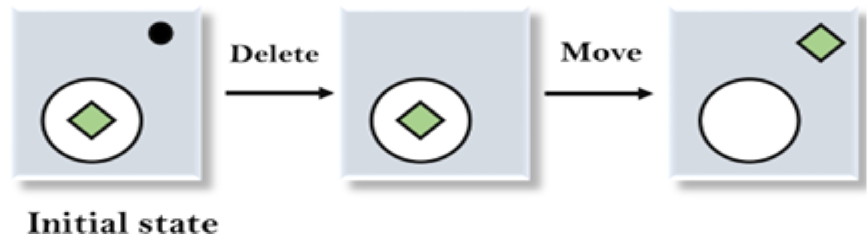
1. Evaluating the initial state: In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.



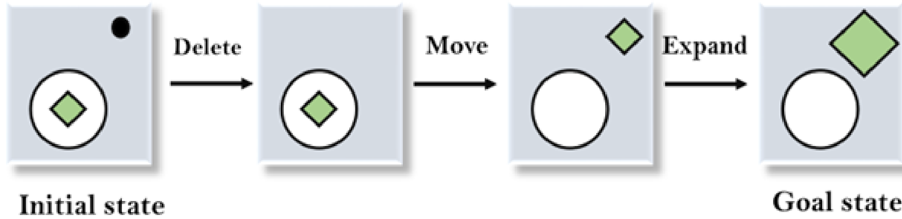
2. Applying Delete operator: As we can check the first difference is that in goal state there is no dot symbol which is present in the initial state, so, first we will apply the **Delete operator** to remove this dot.



3. Applying Move Operator: After applying the Delete operator, the new state occurs which we will again compare with goal state. After comparing these states, there is another difference that is the square is outside the circle, so, we will apply the **Move Operator**.



4. Applying Expand Operator: Now a new state is generated in the third step, and we will compare this state with the goal state. After comparing the states there is still one difference which is the size of the square, so, we will apply **Expand operator**, and finally, it will generate the goal state.



Check your Progress-6

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

1. Define Means End Analysis

…………………………………………………………………

……………………………………………………………….

2. How to evaluate the initial state?

…………………………………………………………………

…………………………………………………………………

3. What is meant by operator subgoaling?

………………………………………………………………..

………………………………………………………………..

10.8 ROBOT PROBLEM SOLVING AS A PRODUCTION SYSTEM

The best automation solutions are smart, simple, innovative, and fully leverage technology to create significant business efficiencies and provide great benefits to employees and customers. Today, Universal Robots collaborative robotic arms are being used to solve business problems with creative concepts and innovative solutions that can be integrated into all types and sizes of production. Collaborative robots – cobots, are lightweight and compact, making the process of deploying and redeploying cobots to different tasks very convenient. There is often no need for additional safety equipment (subject to risk assessment) and combined with an intuitive programming interface, a simple pick and place task can be set up and implemented in less than one hour. However, not every business needs a straightforward pick and place application like this. If your production is more complex in nature, there are steps you can take to ensure your automation solution is flexible, collaborative and affordable.

Flexibility

Many companies run small batch productions with ever changing requirements, and it's these companies that can greatly benefit from the flexibility of deploying cobots. A relatively low initial investment and rapid return on investment are critical for an SME to make the decision to implement the first robots into their facility.

- **Minimize customized fixed machinery & traditional automation**

If the proposed solution includes a large amount of inflexible automation in order to complete the task, and the robot only constitutes a very small part of the overall solution, many of the key benefits of a collaborative robot can be lost. The one-time engineering costs for designing all of the additional equipment will be very high, and if you decide to redeploy the robot, it is often not possible to repurpose this fixed machinery. In many instances, this isn't the most cost-effective solution for a business.

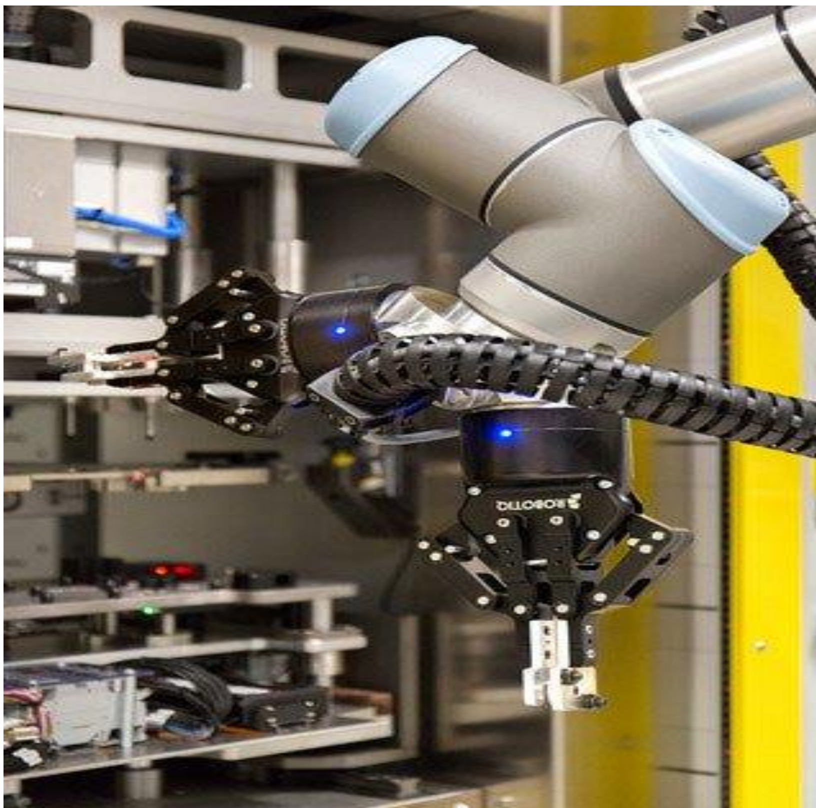
- **Low cost vision systems add flexibility**

Instead of designing a complex feeder system to align incoming products, a low-cost vision solution can allow the robot to identify any changes in position and orientation of products to be picked.

If they're implemented well, they won't result in any reduction in reliability for the system. Modern integrated smart cameras are available for a very affordable price and are easy to integrate, increasing the capabilities of the robot. A vision system can also be redeployed just as easily as the robot to its next task.

- ***An adaptive gripper is a good investment***

While a simple pneumatic chuck may do the job for a current application and cost a little less, additional costs will be incurred if the robot is redeployed, either in designing/producing new jaws or purchasing a new end-effector as the original is not suitable for new tasks. Pneumatics also add complexity to the system with additional components and the requirement for an air supply to be accessible wherever the robot is deployed. By selecting an adaptive gripper the robot can handle a wide range of products, and with many grippers, be controlled in either position or force mode for handling of products with uncertain shapes/sizes.





Collaboration

When it comes to space, cost and scope of applications, cobots have a big advantage because they can operate without safety fencing (subject to risk assessment) and work alongside workers safely. This opens up new application possibilities, where people aren't just allowed, they are required to be in the robot workspace.

- **Active collaborative applications**

While in many cases a robot takes over a repetitive task from a worker allowing them to focus on more engaging work, it's also increasingly common to have a robot working directly alongside a skilled worker on a complex task to increase their productivity. Where it is not possible to completely automate a task, the robot can be used as a smart tool to assist the worker, with increased product output and quality; this can still be a financially viable option. In certain artisanal industries, having the worker involved can retain individuality and increase the value of the product.

- **Understanding safety**

A comprehensive understanding of the robots, the safety system, and risks are essential for the success of the deployment.

Easy programming of smart applications

The Polyscope programming environment is very easy to learn, and also offers some very useful advanced features to allow for rapid and efficient programming of new applications.

- **Rapid prototyping**

If you can create a program on your cobot in as little as 20 minutes, why wait 10 days for something to be fabricated by an external supplier? Time and cost investment is greatly reduced if you do it yourself, giving your designers and engineers the freedom to try out their ideas without consequences.

- **Multi-staged applications**

If a single robot can handle multiple stages of a process, the return on investment for the customer will obviously be even more attractive. For example, a robot in a plastic injection molding plant can apply labels to containers before packing them into boxes for shipping. A robot in consumer electronics factory can place multiple components into a plastic casing before screwing them into place and closing the casing.



- **Out of the box solutions**

The fewer non-recurring engineering hours that go into a solution, the greater the savings for the customer, with an entirely out of the box solution that can be replicated for multiple customers being the ultimate goal. While two projects will rarely be exactly the same, it’s a good idea to implement a solution so that as much of the work is reusable as possible. Writing the robot program in a modular manner makes a big difference when it comes to reusability. Splitting an application into sub-programs for sub-tasks within the program makes it easy to drop them straight into your next project.

Check your Progress-7

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

1. Define Flexibility

…………………………………………………………………

……………………………………………………………….

2. What is meant by collaboration?

…………………………………………………………………

…………………………………………………………………

3. Write about easy programming of smart applications

…………………………………………………………………

…………………………………………………………………

10.9 Triangle table

Structures called triangle tables were used in connection with the SRI robot Shakey for storing sequences of robot actions. Because the rationale for triangle tables still seems relevant, I have recently elaborated the original concept and have begun to consider how the expanded formalism could be used as a general robot-programming language. The present article describes this new view of triangle tables and how they might be used in a language that supports asynchronous and concurrent action computations. Triangle tables can be thought of as a specialized way of writing production rules.

In our conception of a robot system, the production rules operate by testing a condition on the robot's memory. If the condition is satisfied, the action part of the rule is executed. The action may have an effect on the external world through robot effectors and/or it may alter the memory or the perception system. (It is even possible that the action may change the production rules themselves, although we do not consider such a possibility in this note.) We imagine that the memory is a collection of first-order formulas, typically ground literals. The condition part of the production rule is a logical formula, and the memory test consists of determining whether or not this formula follows from the literals in memory.

We imagine that sensory information affects robot actions by virtue of being recorded as formulas in memory. The perception system converts sensory signals into these formulas. Thus, the production rules do not test the world "directly," but rather are sensitive only to a representation or "model" of the world in memory.

In a special case of such a system, the production rule conditions are all mutually exclusive and exhaustive. That is, the memory and rules are organized so that the memory always satisfies precisely one of the production rules. The system then is readily seen to be a finite-state machine whose states correspond to memory equivalence classes defined by each of the production rule conditions. We shall not limit Ourselves to this special case, however.

Production rule formalisms have several advantages for robot systems. First, they are simple and modular. Changes can be made by adding, deleting or modifying rules without having to change the production rule interpreter. Second, they can have short "sense-act" cycles and are therefore not as blind to changes in the world as are systems committed to lengthy actions that cannot be modified by sensory data. Thus, production rule systems are robust in that undesired effects of an inappropriate or abortive action can be noticed quickly so that corrective action can be initiated.

Syntax

Programs consisting of robot actions are designed to achieve certain sequences of effects—some of which are related to the purposes of the programs, while some merely enable subsequent actions to be executed. Information about the preconditions and effects of each robot action within a larger program and how these actions are interrelated can be conveniently represented in a triangular table. Interestingly, this very table can be used as a representation of the program itself. The syntax and semantics of these tables will make clear what they are and how they are used. A triangle table, of rank N, is an N × N triangular array of cells; each cell may contain a collection of formulas, which latter might contain schema variables. The rows of the array are numbered from the top, starting with row 1; the columns of the array are numbered from the left starting with column 0. Each column except the 0th is headed by an action schema. The schema variables in any action schema are a subset of those formula schema variables present in the cells in the row to the left of that action schema. The schema variables in any formula are a subset of those action schema variables present in the action (if any) heading the column in which that formula appears.

The expressions A^B(x),D(y),¬C(x),E^F,G(y),H, I are formula schemas; the expressions a1(x), a2(y), a3(y) are action schemas.

Semantics

Before discussing how a triangle table is used as a program of robot actions, we first explain what the formulas in a table are intended to mean and why they are placed as they are. The conjunction of all the formulas in the row to the left of an action is a precondition for executing that action. More precisely, if an instance of this conjunction follows from the formulas in memory, the corresponding instance of the action can be executed. The conjunction of the formulas in the column immediately beneath an action is the presumed effect (on memory) of that action. More precisely, after an instance of that action is executed, then the corresponding instance of the formulas in that column can be presumed to follow from memory. The only formulas underneath an action in a triangle table are those effects that are also either preconditions of actions heading higher-numbered columns or effects in the last row of the table. The formulas representing the effects of actions are distributed among the column cells in such a way that those that are preconditions of other actions are in a cell to the left of that action.

for example, E^F ^G(y) is a precondition for a3(y) in the sense that, if an instance of the precondition is true, then the corresponding instance of a3 can be executed. Instances of the formula ¬C(x) are effects of corresponding instances of the action a1(x); E ^ F is an additional effect of all instances of a1(x). H and I are effects of a2 and a3, respectively, which are not preconditions of any actions named.

It is intended that a triangle table program will be typically executed by executing actions in the sequence {a1, a2, . . . , }, where ai is the action heading the ith column. (We say typically because, as we shall see, one of the features of triangle tables is that we can deviate from this sequence when appropriate.) The intention is that preconditions of subsequent actions might be among the effects achieved by ai. Those effects that are not accomplished solely to achieve preconditions of subsequent actions

are represented by formulas in the last row of the table. These are the final effects of the table. Preconditions of actions that are not realized by any actions in the table must hold before the table is executed; these preconditions are represented by formulas in the 0th column of the table. These are the initial preconditions of the table. To be listed in the table, any effect F needed by action aj and provided by a previous action ai must survive the intervening actions. An effect represented by a formula in the last row of the table survives all actions subsequent to the one that achieved it.

The conjunction of the formula schemas in the rectangular subarray consisting of the bottom N −(n−1) rows of the leftmost n columns is called the nth kernel of the table. The second kernel of the table in Figure 2 is D(y) ^ ¬C(x) ^E ^ F. The third kernel is E ^ F ^ G(y) ^ H. Each kernel can be thought of as the precondition that must hold for a certain sequence of actions to be executable and for the effects of the table to be achieved. Instances of the ith kernel are preconditions for corresponding instances of the action schema sequence {ai, . . . , aN} to be executable and to achieve the effects that appear in the last row of the table. If an instance of the Nth kernel follows from memory, then without executing any actions, a corresponding instance of the table's effects from memory also follows. The first kernel, which is the precondition of the entire table, must have a true instance for the corresponding instance of the entire sequence of actions to be executable and to achieve the table's effects.

Executing Triangle Tables

To illustrate how a triangle table program works, we consider a simple, if fanciful, example. Consider a robot with the following primitive actions:

goto(x)—the robot goes to place x

pickup(x)—the robot picks up object x

wait—the robot waits and does nothing

hand(y, x)—the robot hands object x to person y

With these actions, the intended effect of the triangle table in Figure 3 is HAV E(y, x)—

person y has object x. In other words, this is a triangle table for delivering objects

AT(R, x)—The robot R is at place x
H(y)—Person y is at work today
HAV E(y, x)—Agent y has object x
p(x)—The present location of object or person x
o(y)—The usual location (say, the office) of person y
R—The robot

We assume that the memory has information in it that allows the robot to determine where it, objects, and people are located. (Or at least it has information that allows it to establish the truth of the predicates in the table when they are in fact true.) The intended sequence of actions prescribed by this table is for the robot to go to the object's location, pick the object up, go to the office where the recipient usually is, wait for the recipient (if necessary), and then hand over the object. The only precondition for this table is that the intended recipient be at work today.

A triangle table is called when it is desired to achieve an instance of its effect. Suppose we want a robot to achieve the condition HAV E(JOHN, PAY CHECK). We can do this by calling the instance deliver (JOHN, PAY CHECK). Even this rather simple example illustrates two important features of this style of programming. One is opportunism. If, for example, the memory ever indicates that HAVE (JOHN, PAY CHECK) is true (for whatever reason), the last kernel will be the active one.

Check your Progress-8

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

1. Define triangle table

…………………………………………………………………

…………………………………………………………………..

2. What is meant by semantic?

…………………………………………………………………

…………………………………………………………………

10.10 Robot Learning

Robot learning is a research field at the intersection of machine learning and robotics. It studies techniques allowing a robot to acquire novel skills or adapt to its environment through learning algorithms. The embodiment of the robot, situated in a physical embedding, provides at the same time specific difficulties (e.g. high-dimensionality, real time constraints for collecting data and learning) and opportunities for guiding the learning process (e.g. sensorimotor synergies, motor primitives).

Robot learning can be closely related to adaptive control, reinforcement learning as well as developmental robotics which considers the problem of autonomous lifelong acquisition of repertoires of skills. While machine learning is frequently used by computer vision algorithms employed in the context of robotics, these applications are usually not referred to as "robot learning".

Adaptive control is the control method used by a controller which must adapt to a controlled system with parameters which vary, or are initially uncertain. For example, as an aircraft flies, its mass will slowly decrease as a result of fuel consumption; a control law is needed that adapts itself to such changing conditions. Adaptive control is different from robust control in that it does not need *a priori* information about the bounds on these uncertain or time-varying parameters; robust control guarantees that if the changes are within given bounds the control law need not be changed, while adaptive control is concerned with control law changing itself.

Parameter Estimation

The foundation of adaptive control is parameter estimation, which is a branch of system identification. Common methods of estimation include recursive least squares and gradient descent. Both of these methods provide update laws which are used to modify estimates in real time (i.e., as the system operates). Lyapunov stability is used to derive these update laws and show convergence criteria (typically persistent excitation; relaxation of this condition are studied in Concurrent Learning adaptive control). Projection (mathematics) and normalization are commonly used to improve the robustness of estimation algorithms.

Classification of adaptive control techniques

In general, one should distinguish between:

1. Feedforward adaptive control
2. Feedback adaptive control

as well as between

1. Direct methods
2. Indirect methods
3. Hybrid methods

Direct methods are ones wherein the estimated parameters are those directly used in the adaptive controller. In contrast, indirect methods are those in which the estimated parameters are used to calculate required controller parameters.[1] Hybrid methods rely on both estimation of parameters and direct modification of the control law.

There are several broad categories of feedback adaptive control (classification can vary):

- Dual adaptive controllers – based on dual control theory
  - Optimal dual controllers – difficult to design
  - Suboptimal dual controllers

Nondual adaptive controllers

- Adaptive pole placement
- Extremum-seeking controllers
- Iterative learning control
- Gain scheduling
- Model reference adaptive controllers (MRACs) – incorporate a *reference model* defining desired closed loop performance
  - Gradient optimization MRACs – use local rule for adjusting params when performance differs from reference. Ex.: "MIT rule".
  - Stability optimized MRACs
- Model identification adaptive controllers (MIACs) – perform system identification while the system is running
  - Cautious adaptive controllers – use current SI to modify control law, allowing for SI uncertainty
  - Certainty equivalent adaptive controllers – take current SI to be the true system, assume no uncertainty
    - Nonparametric adaptive controllers
    - Parametric adaptive controllers
      - Explicit parameter adaptive controllers
      - Implicit parameter adaptive controllers

Multiple models – Use large number of models, which are distributed in the region of uncertainty, and based on the responses of the plant and the models. One model is chosen at every instant, which is closest to the plant according to some metric.

Reinforcement learning (**RL**) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

It differs from supervised learning in that labelled input/output pairs need not be presented, and sub-optimal actions need not be explicitly corrected. Instead the focus is finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

Developmental robotics (**DevRob**),

sometimes called epigenetic robotics, is a scientific field which aims at studying the developmental mechanisms, architectures and constraints that allow lifelong and open-ended learning of new skills and new knowledge in embodied machines. As in human children, learning is expected to be cumulative and of progressively increasing complexity, and to result from self-exploration of the world in combination with social interaction. The typical methodological approach consists in starting from theories of human and animal development elaborated in fields such as developmental psychology, neuroscience, developmental and evolutionary biology, and linguistics, then to formalize and implement them in robots, sometimes exploring extensions or variants of them. The experimentation of those models in robots allows researchers to confront them with reality, and as a consequence, developmental robotics also provides feedback and novel hypotheses on theories of human and animal development.

Developmental robotics is related to but differs from evolutionary robotics(ER). ER uses populations of robots that evolve over time, whereas DevRob is interested in how the organization of a single robot's control system develops through experience, over time.

DevRob is also related to work done in the domains of robotics and artificial life.

Machine learning (**ML**) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop a conventional algorithm for effectively performing the task.

Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

Computer vision is an interdisciplinary scientific field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.

Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, e.g. in the forms of decisions. Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that can interface with other thought processes and elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

The scientific discipline of computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. The technological discipline of computer vision seeks to apply its theories and models to the construction of computer vision systems.

10.8 Unit-End Exercises

1. List out the types of search algorithms.

2. Elaborate Depth-Limited Search Algorithm:

3. Define rapid prototyping.

4. Explain monkey and banana problem.

5. Describe perceptual learning.

6. Define map learning.

7. Explain about smart applications.

8. Explain Bidirectional search algorithm.

9. Discuss about A* search algorithm.

10. Briefly discuss about robot learning.

Answer to check your progress

1. Define Artificial intelligence in robotics

Robotics is a domain in artificial intelligence that deals with the study of creating intelligent and efficient robots. Robots are the artificial agents acting in real world environment.

2. Describe autonomous robots.

Autonomous robots are self-supported. They use a program that provides them the opportunity to decide the action to perform depending on their surroundings.

Using artificial intelligence these robots often learn new behavior. They start with a short routine and adapt this routine to be more successful in a task they perform.

3. Define State Space Search

State Space Search is a process used in the field of computer science, including artificial intelligence (AI), in which successive configurations or *states* of an instance are considered, with the intention of finding a *goal state* with a desired property.

4. Describe best first search algorithm.

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms.

5. Define Block world

- A flat surface such as a tabletop
- An adequate set of identical blocks which are identified by letters.
- The blocks can be stacked one on one to form towers of apparently unlimited height.
- The stacking is achieved using a robot arm which has fundamental operations and states which can be assessed using logic and combined using logical operations.
- The robot can hold one block at a time and only one block can be moved at a time.

6. Why Use the Blocks world as an example?

The blocks world is chosen because:

- it is sufficiently simple and well behaved.
- easily understood
- yet still provides a good sample environment to study planning:
  - problems can be broken into nearly distinct subproblems
  - we can show how partial solutions need to be combined to form a realistic complete solution.

7. Define path selection.

Many AI problems can be cast as the problem of finding a path in a graph. A graph is made up of nodes and arcs. Arcs are ordered pairs of nodes that can have associated costs.

Suppose we have a set of nodes that we call "start nodes" and a set of nodes that we call "goal nodes", a **solution** is a path from a start node to a goal node.

8. What is meant by frontier?

The **frontier** is a set of paths from a start node (we often identify the path with the node at the end of the path). The nodes at the end of the frontier are outlined in green or blue. Initially the frontier is the set of empty paths from start nodes.

9. Define AND OR graph.

The AND-OR GRAPH (or tree) is useful for representing the solution of problems that can solved by decomposing them into a set of smaller problems, all of which must then be solved. This decomposition, or reduction, generates arcs that we call AND arcs. One AND arc may point to any number of successor nodes, all of which must be solved in order for the arc to point to a solution.

10. Give the algorithm of AND OR Graph

ALGORITHM:

1. Let G be a graph with only starting node INIT.
2. Repeat the followings until INIT is labeled SOLVED or h(INIT) > FUTILITY

Select an unexpanded node from the most promising path from INIT (call it NODE)

Generate successors of NODE. If there are none, set h(NODE) = FUTILITY (i.e., NODE is unsolvable); otherwise for each SUCCESSOR that is not an ancestor of NODE do the following:

i. Add SUCCESSSOR to G.

ii. If SUCCESSOR is a terminal node, label it SOLVED and set h(SUCCESSOR) = 0.

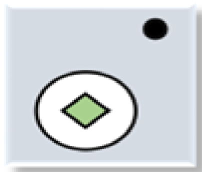
iii. If SUCCESSPR is not a terminal node, compute its h.

11. Define Means End Analysis.

A mixture of the two directions is appropriate for solving a complex and large problem. Such a mixed strategy, make it possible that first to solve the major part of a problem and then go back and solve the small problems arise during combining the big parts of the problem. Such a technique is called **Means-Ends Analysis**.

12. How to evaluate the initial state?

In the first step, we will evaluate the initial state and will compare the initial and Goal state to find the differences between both states.



Initial state

13. What is meant by operator subgoaling?
we create the subproblem of the current state, in which operator can be applied, such type of backward chaining in which operators are selected, and then sub goals are set up to establish the preconditions of the operator is called Operator Subgoaling.

14. Define Flexibility.
Many companies run small batch productions with ever changing requirements, and it's these companies that can greatly benefit from the flexibility of deploying cobots. A relatively low initial investment and rapid return on investment are critical for an SME to make the decision to implement the first robots into their facility.

15. What is meant by collaboration?
Space, cost and scope of applications, cobots have a big advantage because they can operate without safety fencing (subject to risk assessment) and work alongside workers safely. This opens up new application possibilities, where people aren't just allowed, they are required to be in the robot workspace.

16. Write about easy programming of smart applications.

- Rapid prototyping
- Multi-staged applications
- Out of the box solutions

17. **Define triangle table.**

Structures called triangle tables were used in connection with the SRI robot Shakey for storing sequences of robot actions. Because the rationale for triangle tables still seems relevant.

18. What is meant by semantic?

The conjunction of all the formulas in the row to the left of an action is a precondition for executing that action. More precisely, if an instance of this conjunction follows from the formulas in memory, the corresponding instance of the action can be executed.

UNIT – XI AI IN ROBOTICS

11.1 Robot Task planning

11.1.1 Introduction

Robot programming is particularly difficult in the case of complicated tasks requiring complex operations in three-dimensional workspaces where those operations are also coordinated by the information from sensors. And even for relatively simple tasks performed by currently produced industrial robots, the cost involved in their programming can be comparable to the price of the robot itself. Consequently, it is only natural that methods for simplifying robot programming are a priority issue. Treating a robot's operations only in terms of results on the objects of manipulation seems to be one way of solving this problem. For instance, it is easier for a user to define an operation: "place the pivot in the hole" than specify the sequence of manipulator motions required for achieve that result.

To perform the task, specified in this simplified manner, in the workspace, the robot must have information about the objects to be found there, the relations (usually geometrical) between them, the objects it is made of itself, and the admissible actions implying changes in those relations. In other words, a robot must have information about: geometric models (representations) of the workspace, as well as of itself; kinematic and dynamic models (equations) of the manipulator, together with the conditions of their application set, for example, by design limitations, the performances of the actuators used and, in general, by the ambiguity of the task performed.

Aprut from those models, a robot must be fitted with sensors of touch, and vision sensors (usually recording two-dimensional images of the workspace) which make it possible to recognize objects of manipulation during operation planning and updating of the model of the workspace. In the event of changes in the workspace (the introduction of new objects, of obstacles), the information obtained must be transformed into the correct form of the model of the workspace. This process usually involves the reconstruction of a three-dimensional workspace, done on the grounds of the two-dimensional images obtained from vision sensors. Knowledge of the elements listed above enables the robot to: - obtain (by means of vision sensors), and analyze images of the workspace, as well as create its geometric model;

assign and understand the task assigned by a user in natural language- perform that task by generating elementary operations, that is, a finite sequence of manipulator trajectories (expressed as values of the relative rotations and translations of the joints corresponding to a certain trajectory of a gripper motion, with time as a parameter) that realize that task.

Robots capable of performing the above functions are called the third generation robots, According to the defining characteristics defining third generation robots, they meet the basic aim which of the simple programming of the task assigned by a user.

The analysis of all functional elements of the third generation robot is a very broad issue. Only its main function, that of planning the elementary operations required performing a given task, is considered here. The user-robot communication module, the objective of which is syntactic analysis and semantic interpretation of a natural language after converting speech into a language of symbols, is omitted, as are the vision module that analyzes the information obtained from TV cameras and the teaching module used to generalize the working plans that have been generated.

11.1.2TaskPlanning

By virtue of their versatility, robots can be difficult to program, especially for tasks requiring complex motions involving sensory feedback. In order to simplify programming, *task-level* languages exist that specify actions in terms of their effects on objects.

Example: pin A programmer should be able to specify that the robot should put a pin in a hole, without telling it what sequence of operators to use, or having to think about its sensory or motor operators.

Task planning is divided into three phases: modeling, task specification, and manipulator program synthesis.

There are three approaches to specifying the model state:

Using a CAD system to draw the positions of the objects in the desired configuration.

Using the robot itself to specify its configurations and to locate the object features.

Using symbolic spatial relationships between object features (such as `(face1 against face2)`. This is the most common method, but must be converted into numerical form to be used.

One problem is that these configurations may *overconstrain* the state. Symmetry is an example; it does not matter what the orientation of a peg in a hole is. The final state may also not completely specify the operation; for example, it may not say how hard to tighten a bolt.

The three basic kinds of motions are free motion, guarded motion, and compliant motion.

An important part of robot program synthesis should be the inclusion of sensor tests for error detection.

Check your Progress-1

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

1. Define task planning.

…………………………………………………………

………………

2. Describe the approaches of task planning.

……………………………………………………………

11.2 Phases in task planning:

i) Modeling
ii) Task specification, and
iii) Manipulator program synthesis.

Modeling

Modeling of task planning for multirobot system is developed from two parts: task decomposition and task allocation. In the part of task decomposition, the conditions and processes of decomposition are elaborated. In the part of task allocation, the collaboration strategy, the framework of reputation mechanism, and three types of reputations are defined in detail, which include robot individual reputation, robot group reputation, and robot direct reputation.

Manipulator program synthesis.

Robotic task program synthesis embodiments are presented that generally synthesize a robotic task program based on received examples of repositioning tasks. In one implementation, the exemplary repositioning tasks are human demonstrations of object manipulation in an actual or displayed robot workspace. A domain specific language (DSL) designed for object repositioning tasks is employed for the robotic control program. In general, candidate robotic task programs are generated from the example tasks. Each candidate program includes instructions for causing the robot to reposition objects, and represents a different permutation of instructions consistent with the received example tasks.

Robotic Task Program Synthesis

The robotic task program synthesis embodiments described herein generally synthesize a robotic control program based on received examples of repositioning tasks (e.g. sorting, kitting, or packaging). In one implementation, the exemplary repositioning tasks are human demonstrations of object manipulation in an actual or displayed robot workspace.

A domain specific language (DSL) designed for object repositioning tasks is employed for the robotic control program. This DSL is generally a generic stack-based programming language that can be used to specify a wide range of object repositioning tasks. As will be described in more detail in the sections to follow, a DSL program is found that is consistent with the received examples, and which generalizes to accomplish tasks in a robot workspace even with a different arrangement of objects than were available during training. In addition, this program is synthesized automatically and does not require any programming on the part of the user.

Robotic Task Program Synthesis Processes

In view of the foregoing, one general implementation of the robotic task program synthesis embodiments described herein is accomplished by using a computing device to perform the following process actions. Referring to FIG. 2, one or more example tasks are received (process action **200**). Each example task includes a scene-reposition pair. The scene portion of a scene-reposition pair includes a collection of objects characterized by their orientation, location and perceptual object characteristics. This represents a starting configuration of the objects in a workspace associated with a robot. The reposition portion of a scene-reposition pair includes repositioning data for one or more of the objects. The repositioning data for an object includes a destination orientation, or destination location, or both. This data is indicative of a task that it is desired for the robot to perform on various objects in the workspace.

An object in the foregoing process is characterized by a set of named properties—namely orientation, location and perceptual object characteristics. With regard to the orientation and location, in one implementation, this pose information is defined by Cartesian coordinates x and y, and a rotation angle θ, i.e., (x, y, θ). This pose scheme assumes a view of the robot workspace from above looking down. It is noted that the coordinates and angle of a pose can be relative to a global coordinate system, or relative to some other location or orientation, or both.

Check your Progress-2

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

1. What are the phases in task planning?

………………………………………………………

…………………………………………………………
2. Define Modeling.
…………………………………………………………

…………………………………………………………

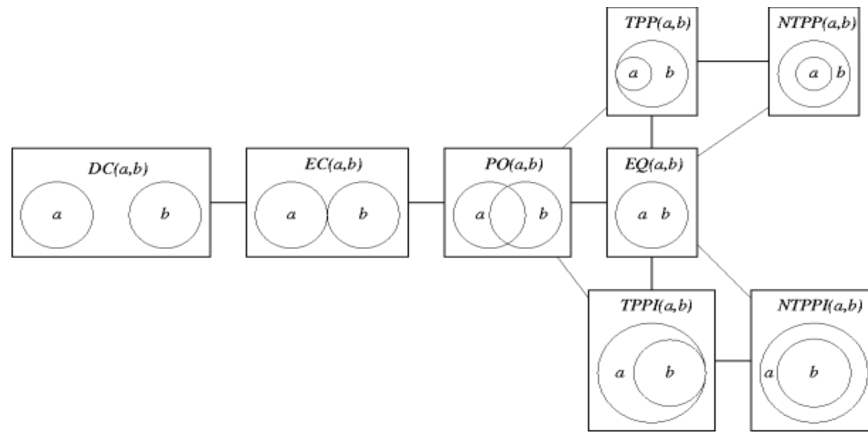
11.3 Symbolic Spatial Relationship

Spatial Knowledge Representation

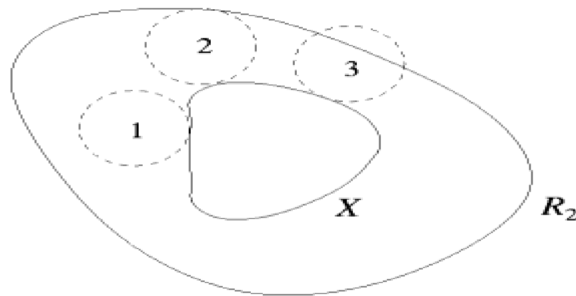
A number of reasons for this may be advanced, but two in particular seem paramount. First, because space has three dimensions, whereas time has only one, it affords a far greater variety of possible structures which a representational system has to handle. One has only to think here of the concept of *shape*, which is of importance in many spatial reasoning contexts; the variety of possible shapes in two dimensions, let alone three, already presents a formidable array of problems to anyone seeking to systematise in a tractable way the processes of representing and reasoning with spatial knowledge. In time, by contrast, the analogous concept to shape (of an interval or event) is virtually empty. The second reason for the delayed development of spatial KR has a rather specific origin in the 'poverty conjecture' of Forbus et al. (1987), that 'there is no problem-independent, purely qualitative representation of space or shape' (Forbus 1995). Forbus adduces this as an explanation for the fact that we humans are so reliant on diagrams and other perceptual representations for our spatial reasoning, since these can capture metric properties absent from a purely qualitative representation, and it is clear that many spatial reasoning tasks cannot be accomplished without access to at least some metric information..

Relation	Symbol	Meaning
R_1 is *disconnected* from R_2	DC	R_1 and R_2 are not connected.
R_1 is *part* of R_2	P	Every region connected to R_1 is connected to R_2.
R_1 *overlaps* R_2	O	Some region is part of both R_1 and R_2
R_1 is *discrete* from R_2	DR	R_1 does not overlap R_2
R_1 is *externally connected* to R_2	EC	R_1 and R_2 are connected but do not overlap
R_1 *partially overlaps* R_2	PO	R_1 overlaps R_2 but neither is part of the other
R_1 is *equal* to R_2	EQ	Each of R_1 and R_2 is part of the other
R_1 is a *proper part* of R_2	PP	R_1 is part of R_2 but not equal to it
R_1 is a *tangential proper part* of R_2	TPP	R_1 is a proper part of R_2 and some region is EC to both.
R_1 is a *non-tangential proper part* of R_2	NTPP	R_1 is a proper part of R_2 but not a TPP

Some of these relations are *symmetric*, i.e., if the relation holds between R_1 and R_2 then it automatically holds between R_2 and R_1 as well (in other words, the relation holds of the two regions without reference to the order in which they are considered): these are C, DC, DR, O, PO, EC, and EQ. The remaining relations, namely P, PP, TPP, and NTPP, are not symmetric, so they can hold between two regions taken in one order without holding between the same two regions taken in the opposite order;[8] each of these non-symmetric relations has an *inverse*, represented as PI, PPI, TPPI, NTPPI, such that if R_1 and R_2 are PP, then R_2 and R_1 are PPI, and so on. Amongst these 15 relations, eight are singled out as forming a JEPD set analogous to the 13 relations of the Interval Calculus, namely DC, EC, PO, EQ, RPP, NTPP, TPPI, and NTPPI. These form the system RCC8, and Fig. 4 shows the well-known Conceptual Neighbourhood Diagram for these relations.



As with the Interval Calculus, a composition table can be made for the RCC8 relations, which can be used as a basis for qualitative reasoning about location. An example is that if R_1 is externally connected to a non-tangential proper part of R_2 then the relation between the two regions must be either partial overlap or proper part (either tangential or non-tangential). This is illustrated in Fig.



11.4 Obstacle avoidance

- Obstacle avoider robot is the important part of mobile robotics. Obstacle avoidance is task which is used for detecting the presence of object in a path of robot or any vehicle.
- Obstacle avoiding robot is an intelligence device, which is used to protect the robot from any physical damages. It automatically sense and overcome the obstacles on its path.

Obstacle avoidance with the Bug-1 algorithm

Obstacle avoidance is an indispensable behavior in mobile robots. If a robot has no sensors, it's a blind robot. If it has proximity sensors but still it keeps hitting whatever comes in its way, then it's a ''brainless'' robot. Imagine an ant or any insect walking on the ground. The insect is heading towards a particular direction and it encounters an obstacle in its way. What the insect does is, it circumnavigates the obstacle until motion in the initial direction is no more obstructed. The situation is very similar to a car at a cross road. To go straight from a traffic crossing, the car has to circumnavigate the cross-road circle and go straight.

Assumptions and initialization

Essentially, the Bug-1 algorithm formalizes the "common sense" idea of moving towards the goal and going around obstacles. Certain assumptions have to be made while implementing the Bug-1 algorithm, they are:

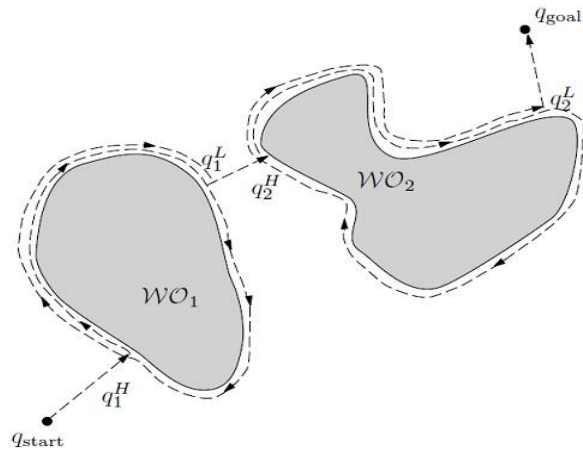
1. The robot is assumed to be a point with perfect positioning (no positioning error)
2. The robot is equipped with a contact sensor that can detect an obstacle boundary if the robot "touches" it.
3. The robot can also measure the distance d(p, q) between any two points p and q.
4. Finally, assume that the workspace is bounded. That is, we're not working in infinite space. We assume the following symbols:
5. q_{start}: start point
6. q_{goal}: target point Let $q_L^0 = q_{start}$ m-line - line segment that connects q_L^i to q_{goal}. Initially i = 0

The Bug-1 Algorithm

The Bug1 algorithm exhibits two behaviors:

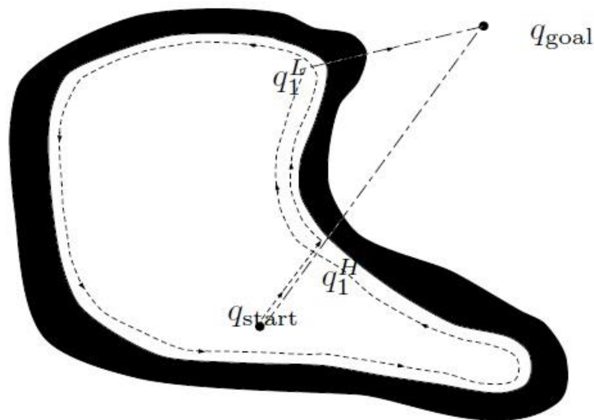
- Motion to goal
- Boundary following

During motion-to-goal, the robot moves along the m-line toward q_{goal} until it either encounters the goal or an obstacle. If the robot encounters an obstacle, let q_{H1} be the point where the robot first encounters an obstacle and call this point a hit point. The robot then circumnavigates the obstacle until it returns to q_{H1}.



Then, the robot determines the closest point to the goal on the perimeter of the obstacle and traverses to this point. This point is called a leave point and is labeled q_{L1}. From q_{L1}, the robot heads straight toward the goal again, i.e., it re-invokes the motion-to-goal behavior.

If the line that connects q_{L1} and the goal intersects the current obstacle, then there is no path to the goal; note that this intersection would occur immediately "after" leaving q_{L1} .



Otherwise, the index i is incremented and this procedure is then repeated for q_{Li} and q_{Hi} until the goal is reached or the planner determines that the robot cannot reach the goal. Finally, if the line to the goal "grazes" an obstacle, the robot need not invoke a boundary following behavior, but rather continues onward towards the goal.

Check your Progress-3

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

1. Define Obstacle avoidance

………………………………………………………………
…

………………………………………………………………..

2. Describe the two behaviors of bug-1 algorithm

11.5 Graph Planning

Graphplan is an algorithm for automated planning. Graphplan takes as input a planning problem expressed in STRIPS and produces, if one is possible, a sequence of operations for reaching a goal state.

The name *graph*plan is due to the use of a novel *planning graph*, to reduce the amount of search needed to find the solution from straightforward exploration of the *state space graph*.

In the *state space graph*:

- the nodes are possible states,
- And the edges indicate reachability through a certain action.

On the contrary, in Graphplan's *planning graph*:

- the nodes are actions and atomic facts, arranged into alternate levels,
- and the edges are of two kinds:
  1. from an atomic fact to the actions for which it is a condition,
  2. From an action to the atomic facts it makes true or false.

The first level contains true atomic facts identifying the initial state.

How to build the planning graph

1. Start from S_0

2. i = 0

3. Find all the actions of A_{i+1} applicable in Si given the mutexes

4. Compute the mutexes between the actions of A_{i+1}

5. Compute the literals reachable in S_{i+1}

6. Compute the mutexes in S_{i+1}

7. If S_{i+1}6= S_i , then increment i by 1 and go to 3

11.5 .1 Mutexes

- A mutex between two actions indicates that it is impossible to perform these actions in parallel.
- A mutex between two literals indicates that it is impossible to have these both literals true at this stage.

How to compute mutexes

Actions

- Inconsistent effects: two actions that lead to inconsistent effects
- Interference: an effect of the first action negates the precondition of the other action
- Competing needs: a precondition of the first action is mutually exclusive with a precondition of the second action.

Literals

- one literal is the negation of the other one
- Inconsistensy support: each pair of action archieving the two literals are mutually exclusive.

11.5.2 Planning graph for heuristic search

- Using the planning graph to estimate the number of actions to reach a goal
- If a literal does not appear in the planning graph, then there is no plan that achieve this literal
- $h = \infty$

Possible heuristics

max-level: take the maximum level where any literal of the goal first appears

- admissible

level-sum: take the sum of the levels where any literal of the goal first appears

- not admissible, but generally efficient (specially for independant subplans)

set-level: take the minimum level where all the literals of the goal appear and are free of mutex

- admissible

11.5.3 Graph plan

function GRAPHPLAN(Problem)

graph ← INITIAL-PLANNING-GRAPH(Problem)

goals ← GOALS(Problem)

loop

if goals all non-mutex in last level of graph then

solution ←

EXTRACT-SOLUTION(graph, goals, LENGTH(graph))

if solution 6= failure then

return solution

else if NO-SOLUTION-POSSIBLE(graph) then

return failure

end if end if

graph ← EXPAND-GRAPH(graph, problem)

end loop

Check your Progress- 4

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

1. Define Graph planning.

……………………………………………………………………

…………………………………………………………………...

2. Define set-level.

……………………………………………………………………

…………………………………………………………………..

3. Describe literals

……………………………………………………………………

11. 5 Unit-End Exercises

1. Explain task planning
2. List out the phases in task planning.
3. Discuss about graph planning.
4. Define obstacle avoidance.
5. Explain how to build the planning graph?
6. Describe mutexes. Explain how to compute mutexes.
7. Define possible heuristics.

Answer to check your progress

1. Define task planning.

By virtue of their versatility, robots can be difficult to program, especially for tasks requiring complex motions involving sensory feedback. In order to simplify programming, *task-level* languages exist that specify actions in terms of their effects on objects.

2. Describe the approaches of task planning.

Using a CAD system to draw the positions of the objects in the desired configuration.

Using the robot itself to specify its configurations and to locate the object features.

3. What are the phases in task planning?

i) Modeling
ii) Task specification, and
iii) Manipulator program synthesis.

4. Define Modeling.

Modeling of task planning for multirobot system is developed from two parts: task decomposition and task allocation. In the part of task decomposition, the conditions and processes of decomposition are elaborated.

5. Define Obstacle Avoidance.

Obstacle avoider robot is the important part of mobile robotics. Obstacle avoidance is task which is used for detecting the presence of object in a path of robot or any vehicle.

6. Describe the two behaviors of bug-1 algorithm.

The Bug1 algorithm exhibits two behaviors:

i) Motion to goal
ii) Boundary following

7. Define graph planning.

Graphplan is an algorithm for automated planning. Graphplan takes as input a planning problem expressed in STRIPS and produces, if one is possible, a sequence of operations for reaching a goal state.

8. Define set-level.

set-level: take the minimum level where all the literals of the goal appear and are free of mutex

- admissible

9.Describe literals.

- one literal is the negation of the other one

Inconsistensy support: each pair of action archieving the two literals are mutually exclusive.

11.6 Suggested Reading

---

1. Stuart Russell and Peter Norvig, Artificial Intelligence A Modern Approach, Second Edition Prentice Hall Series (2003).

2. https://www.slideshare.net/ManjeetKamboj/monkey-banana-problem-in-ai

3. http://aimaterials.blogspot.com/p/and-or-graph.html

4. https://faculty.nps.edu/ncrowe/book/chap11.html

5. https://blog.universal-robots.com/solving-complex-problems-with-innovative-concepts-and-robotic-solutions

6. https://www.hindawi.com/journals/tswj/2014/818701/

7. http://www.grastien.net/ban/teaching/06-planning5.pdf

8. https://www.javatpoint.com/obstacle-avoider-robot

9. https://artint.info/tutorials/search/search_1.html

10. http://aimaterials.blogspot.com/p/planning.html

11.https://www.cs.utexas.edu/~mooney/cs343/slide-handouts/planning.4.pdf

12. https://users.cs.cf.ac.uk/Dave.Marshall/AI2/node116.html

UNIT - XII MACHINE VISION

Structure

12.1 Introduction

Machine vision is the automatic extraction of information from digital images for process or quality control. Most manufacturers use automated machine vision instead of human inspectors because it is better suited to repetitive inspection tasks.

It is faster, more objective, and works continuously. A process involving image based automatic inspection, process control and robot guidance in the industry.

Machine vision system inspection consists of narrowly defined tasks such as counting objects on a conveyor, reading serial numbers, and searching for surface defects.

Manufacturers often prefer machine vision systems for visual inspections that require high speed, high magnification, around-the-clock operation, and/or repeatability of measurements.

Machine vision is the substitution of the human visual sense and judgment capabilities with a video camera and computer to perform an inspection task.

It is the automatic acquisition and analysis of images to obtain desired data for controlling or evaluating a specific part or activity.

Key Points:

• Automated/Non-Contact

• Acquisition

• Analysis

• Data

Machine vision systems rely on digital sensors protected inside industrial cameras with specialized optics to acquire images, so that computer hardware and software can process, analyze, and measure various characteristics for decision making.

As an example, consider a fill-level inspection system at a brewery (Figure 1). Each bottle of beer passes through an inspection sensor, which triggers a vision system to flash a strobe light and take a picture of the bottle.

After acquiring the image and storing it in memory, vision software processes or analyzes it and issues a pass-fail response based on the fill level of the bottle.

If the system detects an improperly filled bottle—a fail—it signals a diverter to reject the bottle. An operator can view rejected bottles and ongoing process statistics on a display.

Machine vision systems can also perform objective measurements, such as determining a spark plug gap or providing location information that guides a robot to align parts in a manufacturing process.

Figure 2 shows examples of how machine vision systems can be used to pass or fail oil filters (right) and measure the width of a center tab on a bracket (left).

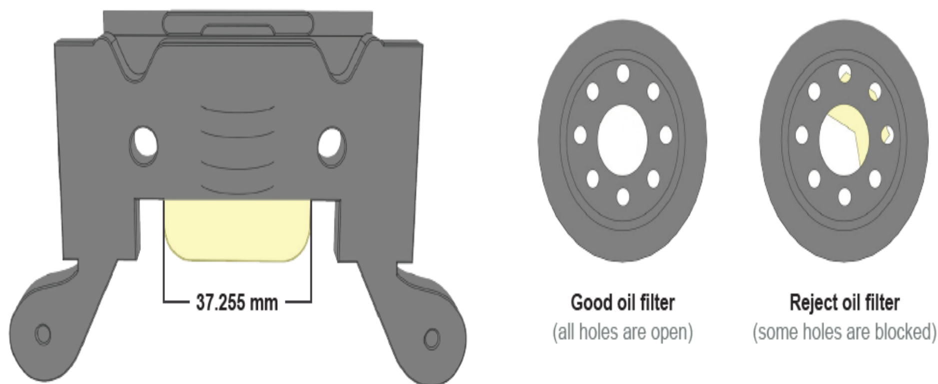


Figure 2. Machine vision systems can process real-time measurements and inspections on the production line, such as a machined bracket (left) or oil filters (right).

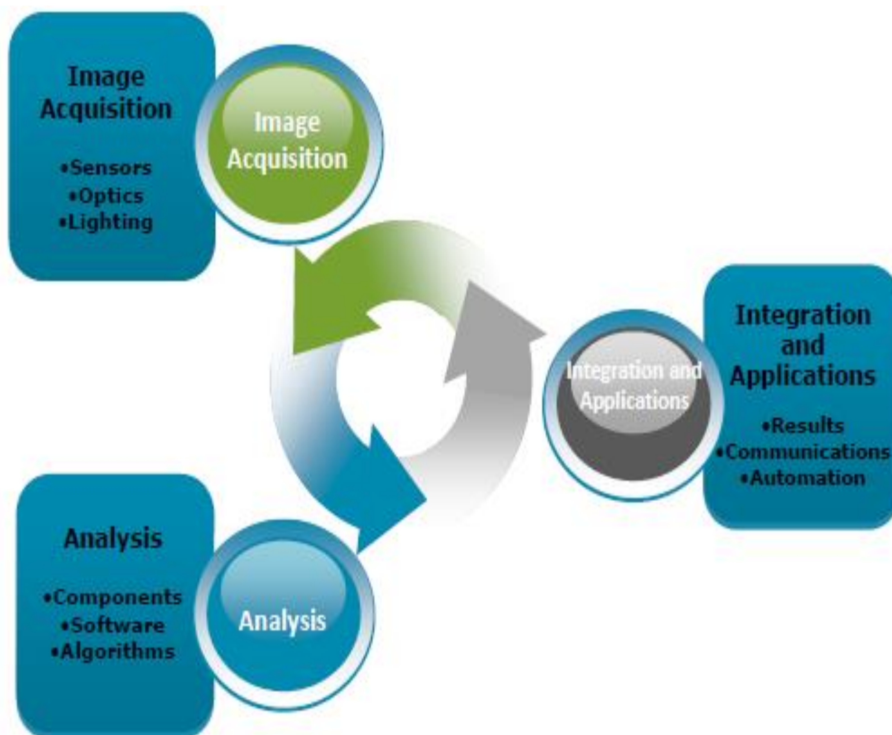


Figure 3: Basic works of Machine Vision

12.1.1 Machine Vision Basics

Machine vision encompasses all industrial and non-industrial applications in which a combination of hardware and software provide operational guidance to devices in the execution of their functions based on the capture and processing of images.

Machine vision helps solve complex industrial tasks reliably and consistently. Machine vision systems rely on digital sensors protected inside industrial cameras with specialized optics to acquire images, so that computer hardware and software can process, analyze, and measure various characteristics for decision making.

Machine vision systems can also perform objective measurements, such as determining a spark plug gap or providing location information that guides a robot to align parts in a manufacturing process.

12.1.2 Benefits of Machine Vision

Where human vision is best for qualitative interpretation of a complex, unstructured scene, machine vision excels at quantitative measurement of a structured scene because of its speed, accuracy, and repeatability.

Machine vision prevents part damage and eliminates the maintenance time and costs associated with wear and tear on mechanical components.

Machine vision brings additional safety and operational benefits by reducing human involvement in a manufacturing process.

Moreover, it prevents human contamination of clean rooms and protects human workers from hazardous environments.

12.1.3 Machine vision helps meet strategic goals

Strategic Goal	Machine Vision Applications
Higher quality	Inspection, measurement, gauging, and assembly verification
Increased productivity	Repetitive tasks formerly done manually are now done by Machine Vision System
Production flexibility	Measurement and gauging / Robot guidance / Prior operation verification
Less machine downtime and reduced setup time	Changeovers programmed in advance
More complete information and tighter process control	Manual tasks can now provide computer data feedback
Lower capital equipment costs	Adding vision to a machine improves its performance, avoids obsolescence
Lower production costs	One vision system vs. many people / Detection of flaws early in the process
Scrap rate reduction	Inspection, measurement, and gauging
Inventory control	Optical Character Recognition and identification
Reduced floor space	Vision system vs. operator

12.1.4 Machine Vision Applications

Typically the first step in any machine vision application, whether the simplest assembly verification or a complex 3D robotic bin-picking, is for pattern matching technology to find the object or feature of interest within the camera’s field of view.

Locating the object of interest often determines success or failure. If the pattern matching software tools cannot precisely locate the part within the image, then it cannot guide, identify, inspect, count, or measure the part.

1. **Guidance**

Guidance may be done for several reasons. First, machine vision systems can locate the position and orientation of a part, compare it to a specified tolerance, and ensure it’s at the correct angle to verify proper assembly.

Next, guidance can be used to report the location and orientation of a part in 2D or 3D space to a robot or machine controller, allowing the robot to locate the part or the machine to align the part.

Machine vision guidance achieves far greater speed and accuracy than manual positioning in tasks such as arranging parts on or off pallets, packaging parts off a conveyor belt, finding and aligning parts for assembly with other components, placing parts on a work shelf, or removing parts from bins.

Guidance can also be used for alignment to other machine vision tools. This is a very powerful feature of machine vision because parts may be presented to the camera in unknown orientations during production.

By locating the part and then aligning the other machine vision tools to it, machine vision enables automatic tool fixturing.

This involves locating key features on a part to enable precise positioning of caliper, blob, edge, or other vision software tools so that they correctly interact with the part.

This approach enables manufacturers to build multiple products on the same production line and reduces the need for expensive hard tooling to maintain part position during inspection.

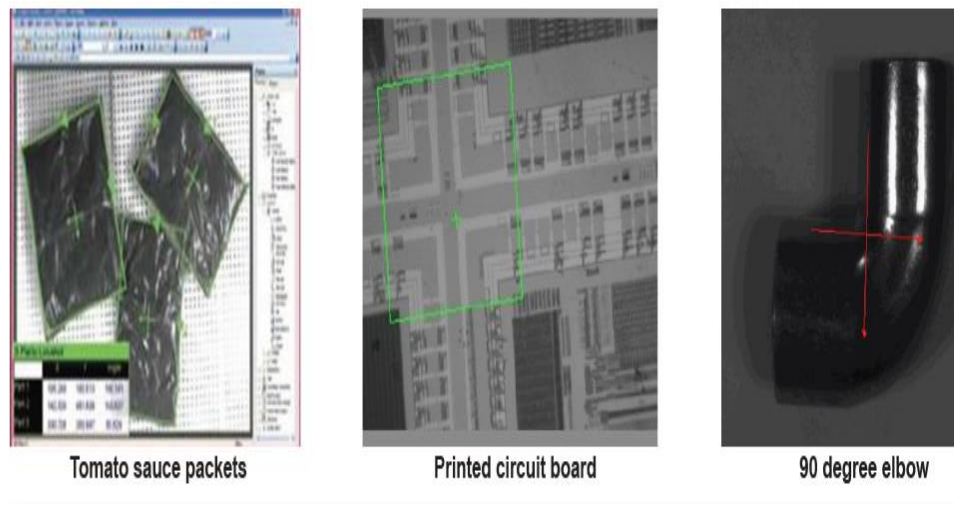


Figure 4. Examples of images used in guidance.

Sometimes guidance requires geometric pattern matching. Pattern matching tools must tolerate large variations in contrast and lighting, as well as changes in scale, rotation, and other factors while finding the part reliably every time.

This is because location information obtained by pattern matching enables the alignment of other machine vision software tools.

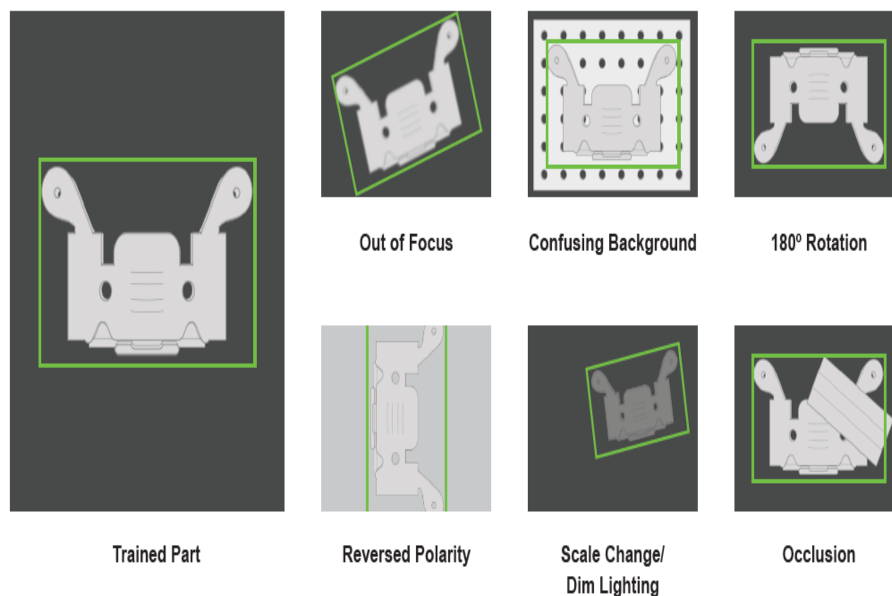


Figure 5. Pattern Matching

2. Identification

A machine vision system for part identification and recognition reads barcodes, data matrix codes, direct part marks (DPM), and characters printed on parts, labels, and packages.

An optical character recognition system reads alphanumeric characters without prior knowledge, whereas an optical character verification system confirms the presence of a character string.

Additionally, machine vision systems can identify parts by locating a unique pattern or identify items based on color, shape, or size.

DPM applications mark a code or character string directly on to the part. Manufacturers in all industries commonly use this technique for error-proofing, enabling efficient containment strategies, monitoring process control and quality-control metrics, and quantifying problematic areas in a plant such as bottlenecks.

Traceability by direct part marking improves asset tracking and part authenticity verification.

It also provides unit level data to drive superior technical support and warranty repair service by documenting the genealogy of the parts in a sub-assembly that make up the finished product.



Figure 6. Identification techniques can range from simple barcode scanning to OCR

Conventional barcodes have gained wide acceptance for retail checkout and inventory control.

Traceability information, however, requires more data than can fit in a standard barcode.

To increase the data capacity, companies developed 2-D codes, such as Data Matrix, which can store more information, including manufacturer, product identification, lot number, and even a unique serial number for virtually any finished good.

3. Gauging

A machine vision system for gauging calculates the distances between two or more points or geometrical locations on an object and determines whether these measurements meet specifications.

If not, the vision system sends a fail signal to the machine controller, triggering a reject mechanism that ejects the object from the line.

In practice, a fixed-mount camera captures images of parts as they pass the camera’s field of view and the system uses software to calculate distances between various points in the image.

Because many machine vision systems can measure object features to within 0.0254 millimeters, they address a number of applications traditionally handled by contact gauging.



Figure 7. Gauging applications can measure part tolerances to within 0.0254 millimeters

4. Inspection

A machine vision system for inspection detects defects, contaminants, functional flaws, and other irregularities in manufactured products.

Examples include inspecting tablets of medicine for flaws, displays to verify icons or confirm pixel presence, or touch screens to measure the level of backlight contrast.

Machine vision can also inspect products for completeness, such as ensuring a match between product and package in the food and pharmaceutical industri es, and checking safety seals, caps, and rings on bottles.

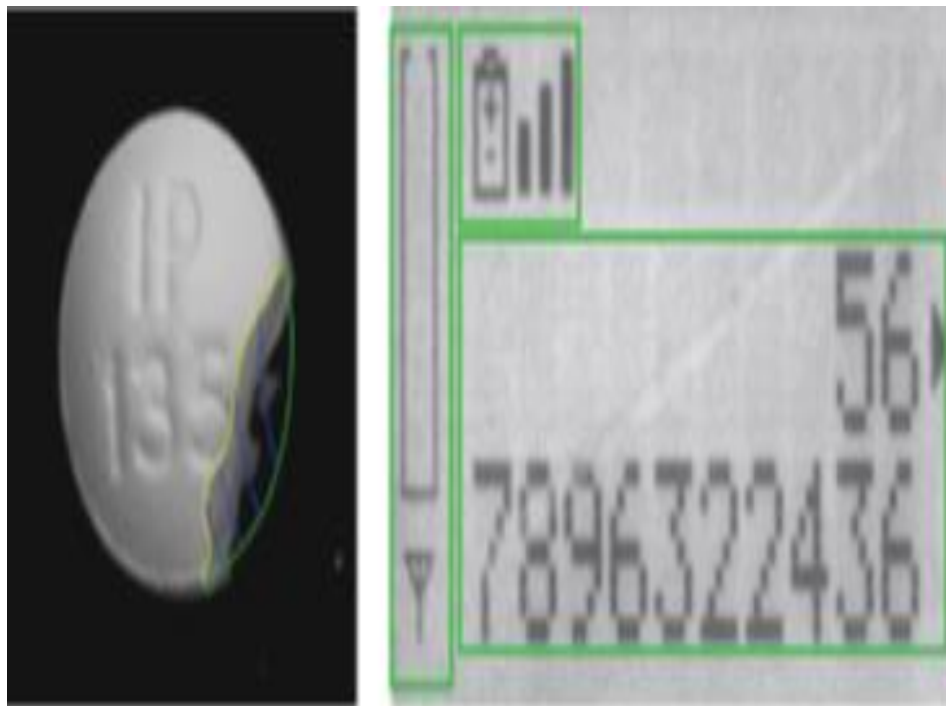


Figure 8. Machine vision systems can detect defects or functional flaws

Check your Progress-1

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Machine Vision

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

ii. Define Machine Vision Benefits

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

iii. Define Guidance

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

iv. Define Gauging

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

v. Define strategic goals

……………………………………………………………………………..

……………………………………………………………………………..

12.2 Functions in a vision system

A machine vision system typically consists of digital cameras and back-end image processing hardware and software. The camera at the front end captures images from the environment or from a focused object and then sends them to the processing system.

A machine vision system primarily enables a computer to recognize and evaluate images. It is similar to voice recognition technology, but uses images instead.

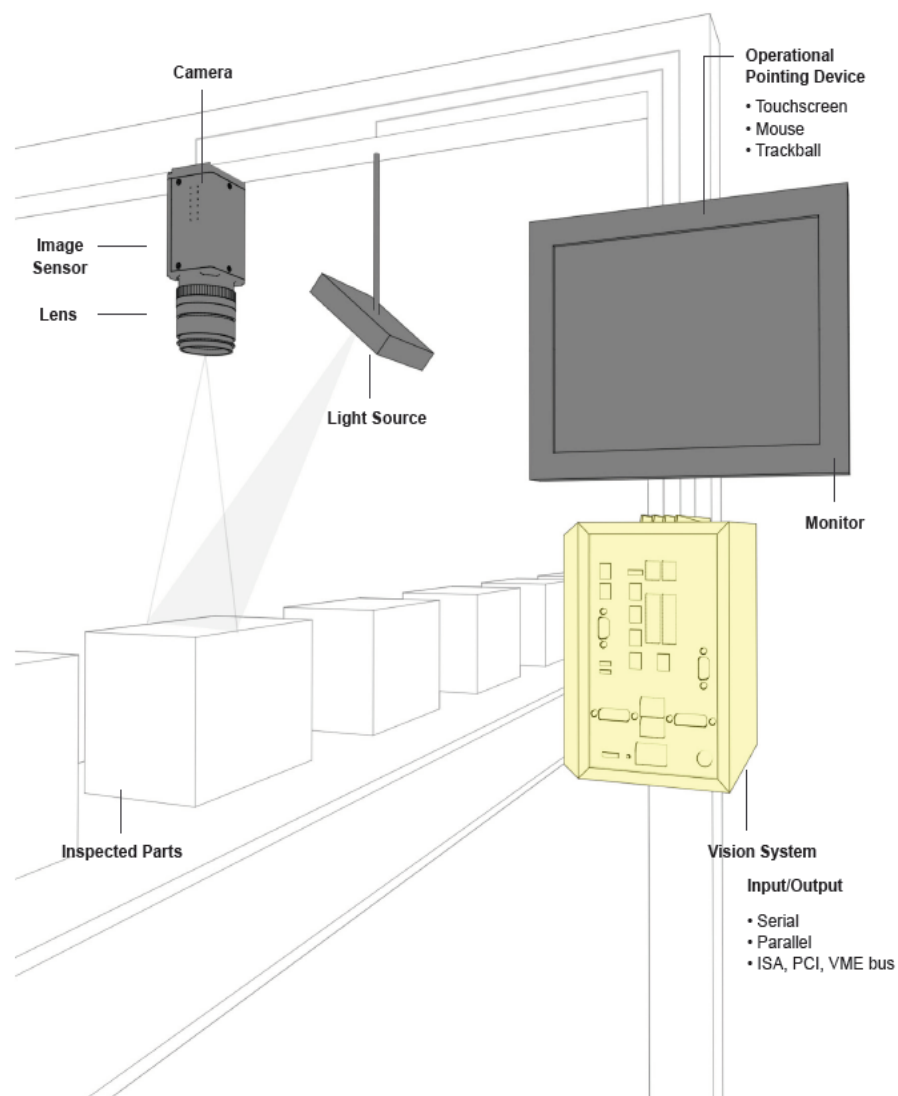


Figure 2: Major Components of a Machine Vision System

The major components of a machine vision system include the lighting, lens, image sensor, vision processing, and communications. Lighting illuminates the part to be inspected allowing its features to stand out so they can be clearly seen by camera.

The lens captures the image and presents it to the sensor in the form of light. The sensor in a machine vision camera converts this light into a digital image which is then sent to the processor for analysis.

Vision processing consists of algorithms that review the image and extract required information, run the necessary inspection, and make a decision. Finally, communication is typically accomplished by either discrete I/O signal or data sent over a serial connection to a device that is logging information or using it.

Most machine vision hardware components, such as lighting modules, sensors, and processors are available commercial off-the-shelf (COTS). Machine vision systems can be assembled from COTS, or purchased as an integrated system with all components in a single device.

A machine vision system including various key components are lighting, lenses, vision sensor, image processing, vision processing and communications.

Machine vision system is a sensor used in the robots for viewing and recognizing an object with the help of a computer. It is mostly used in the industrial robots for inspection purposes. This system is also known as artificial vision or computer vision.

It has several components such as a camera, digital computer, digitizing hardware, and an interface hardware & software.

12.2.1 Three important tasks

The machine vision process includes three important tasks, namely:

1. Sensing & Digitizing Image Data
2. Image Processing & Analysis
3. Applications

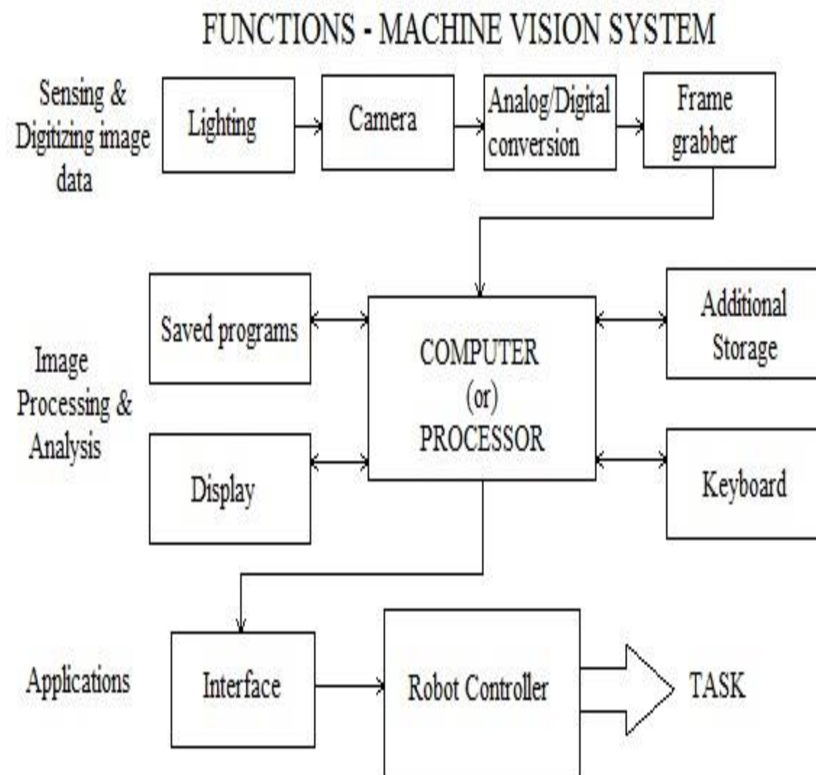


Figure 3: Functions of a Machine Vision System

1. **Sensing & Digitizing Image Data**

A camera is used in the sensing and digitizing tasks for viewing the images. It will make use of special lighting methods for gaining better picture contrast. These images are changed into the digital form, and it is known as the frame of the vision data.

A frame grabber is incorporated for taking digitized image continuously at 30 frames per second. Instead of scene projections, every frame is divided as a matrix. By performing sampling operation on the image, the number of pixels can be identified.

The pixels are generally described by the elements of the matrix. A pixel is decreased to a value for measuring the intensity of light. As a result of this process, the intensity of every pixel is changed into the digital value and stored in the computer's memory.

2. Image Processing & Analysis

In this function, the image interpretation and data reduction processes are done. The threshold of an image frame is developed as a binary image for reducing the data.

The data reduction will help in converting the frame from raw image data to the feature value data.

The feature value data can be calculated via computer programming. This is performed by matching the image descriptors like size and appearance with the previously stored data on the computer.

The image processing and analysis function will be made more effective by training the machine vision system regularly. There are several data collected in the training process like length of perimeter, outer & inner diameter, area, and so on. Here, the camera will be very helpful to identify the match between the computer models and new objects of feature value data.

3. Applications

Some of the important applications of the machine vision system in the robots are,

- Inspection
- Orientation
- Part Identification
- Location

12.2.2 General block diagram of a vision system

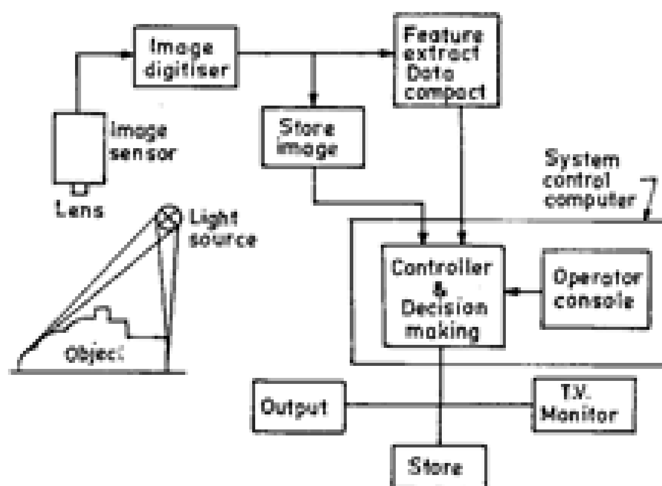


Figure. 4 General block diagram of a vision system

Lighting and presentation of object to be evaluated is very important task in implementing a vision system.

It has great impact on system repeatability, reliability, and accuracy. Lighting source and projection should be chosen such that it accentuates the key features of the object, and gives sharp contrast and detail of the image.

The specular reflections by small angle lighting and other techniques which provide diffused reflection should be avoided.

Image sensor usually comprises of a TV camera, which may be vision TV camera which has greater resolution and low in cost, or it may be solid state camera (charge coupled device CCD or charge injection device CID).

The solid state cameras have greater geometric accuracies, no image lag, and longer life.

Image digitizer is usually a six to eight bit analog to digital A/D converter which is designed to keep up with the flow of video information from camera and store the digitized image in memory.

For simple processing, analog comparator and a computer controller threshold to convert the video information to a binary image is used.

The binary images (having only two values for each pixel) are much simpler and facilitate high speed processing.

However gray scale images contain a great deal more picture information and must be used for complex images with subtle variation of gray level across the image.

Feature extractor/data compactor employs a high speed array processor to provide very high speed processing of the input image data.

To generate a relatively simple feature data set, pattern recognition algorithms need to be implemented.

System control computer communicates with the operator and makes decisions about the part being inspected. These decisions are usually based on some simple operations applied to the feature data set representing the original image.

The output and peripheral devices operate under the control of the system control computer. The output enables the vision system to either control a process or provide action and orientation information two a robot, etc.

Check your Progress-2

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. List out Major Components of a Machine Vision System

…………………………………………………………………………..

…………………………………………………………………………..

…………………………………………………………………………..

ii. Explain the Major functions of a Machine Vision System

…………………………………………………………………………..

…………………………………………………………………………..

…………………………………………………………………………..

…………………………………………………………………….........

iii. List out three important tasks

…………………………………………………………………………..

…………………………………………………………………………..

…………………………………………………………………………..

……………………………………………………………………….....

iv. Important applications of the machine vision system

…………………………………………………………………………..

…………………………………………………………………………..

…………………………………………………………………………..

12.3 Imaging Devices

Diverse Imaging devices are available

- Analog (RS170)
- Digital (GigE Vision, FireWire, Camera Link,USB) interfaces

1. **Lenses**

The lens captures the image and delivers it to the image sensor in the camera.

2. **Image Sensor**

The camera's ability to capture a correctly-illuminated image of the inspected object depends not only on the lens, but also on the image sensor within the camera.

Image sensors typically use a charge coupled device (CCD) or complementary metal oxide semiconductor (CMOS) technology to convert light (photons) to electrical signals (electrons).

Essentially the job of the image sensor is to capture light and convert it to a digital image balancing noise, sensitivity and dynamic range.

3. **Vision Processing**

Processing is the mechanism for extracting information from a digital image and may take place externally in a PC-based system, or internally in a standalone vision system.

4. **Communications**

These items must coordinate and connect to other machine elements quickly and easily.

Typically this is done by either discrete I/O signal or data sent over a serial connection to a device that is logging information or using it.

Discrete I/O points may be connected to a programmable logic controller (PLC), which will use that information to control a work cell or an indicator such as a stack light or directly to a solenoid which might be used to trigger a reject mechanism.

Data communication by a serial connection can be in the form of a conventional RS-232 serial output, or Ethernet.

Some systems employ a higher-level industrial protocol like Ethernet/IP, which may be connected to a device like a monitor or other operator interface to provide an operator interface specific to the application for convenient process monitoring and control.

Check your Progress-3

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Lenses

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

ii. Define Image Sensor

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

iii. Define Vision Processing

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

iv. Define Communications

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

v. Define Diverse Imaging devices

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

12.4 Lighting

Lighting is one key to successful machine vision results. Machine vision systems create images by analyzing the reflected light from an object, not by analyzing the object itself.

A lighting technique involves a light source and its placement with respect to the part and the camera.

A particular lighting technique can enhance an image such that it negates some features and enhances others, by silhouetting a part which obscures surface details to allow measurement of its edges.

The correct lighting must highlight features to be detected relative to background and create repeatable images regardless of part variation.

The incorrect lighting will put the success, reliability, repeatability, ease-of-use of the vision application at risk.

Machine vision cameras and software algorithms cannot make up for inadequate illumination techniques.

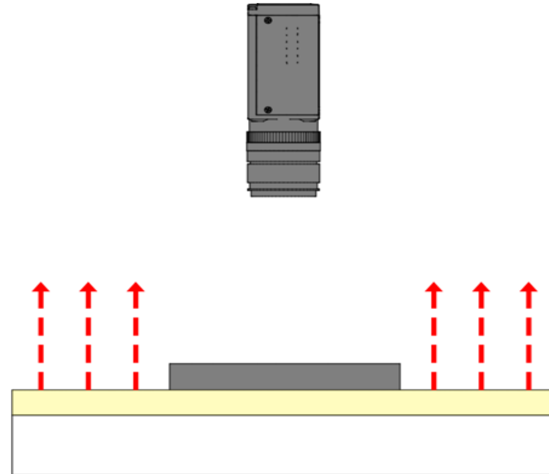
The Illumination for machine vision must be designed for imaging, not human viewing, selection must be made relative to light structure, position, color, diffusion.

The need to know how light works so our light selections are not “hit and miss” guesswork.

Light is both absorbed and reflected to some degree from all surfaces when an object is clear or translucent, light is also transmitted and angle of incidence = angle of reflection.

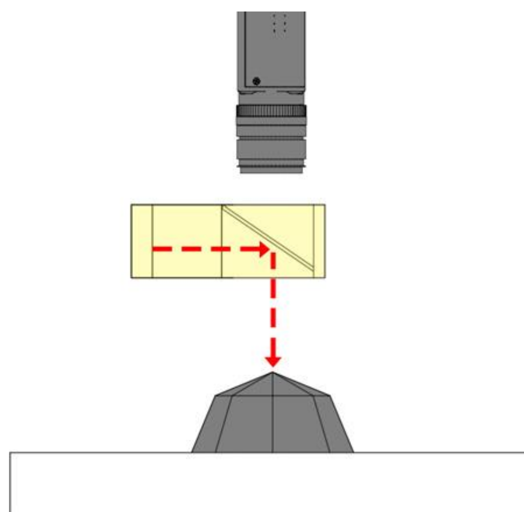
1. **Back lighting**

Back lighting enhances an object's outline for applications that need only external or edge measurements. Back lighting helps detect shapes and makes dimensional measurements more reliable.



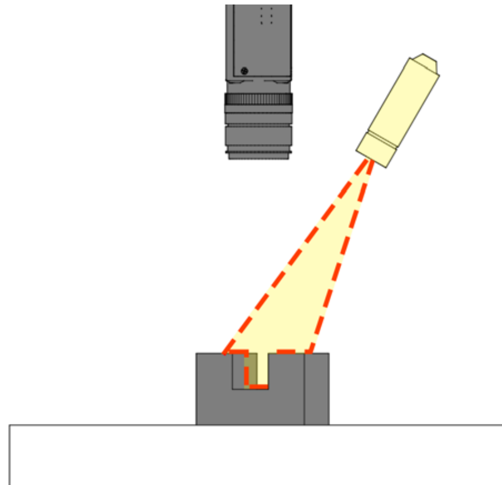
2. **Axial diffuse lighting**

Axial diffuse lighting couples light into the optical path from the side (coaxially). A semitransparent mirror illuminated from the side, casts light downwards on the part. The part reflects the light back to the camera through the semi-transparent mirror resulting in a very evenly illuminated and homogeneous looking image.



3. **Structured light**

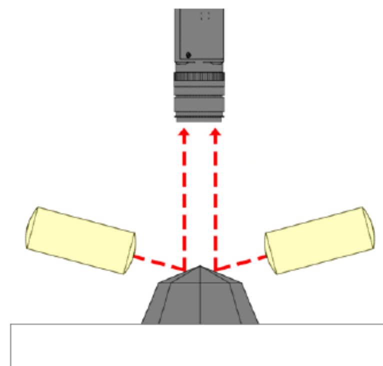
Structured light is the projection of a light pattern (plane, grid, or more complex shape) at a known angle onto an object. It can be very useful for providing contrast-independent surface inspections, acquiring dimensional information and calculating volume.



4. **Dark-field illumination**

Directional lighting more easily reveals surface defects and includes dark-field and bright-field illumination. Dark-field illumination generally preferred for low-contrast applications.

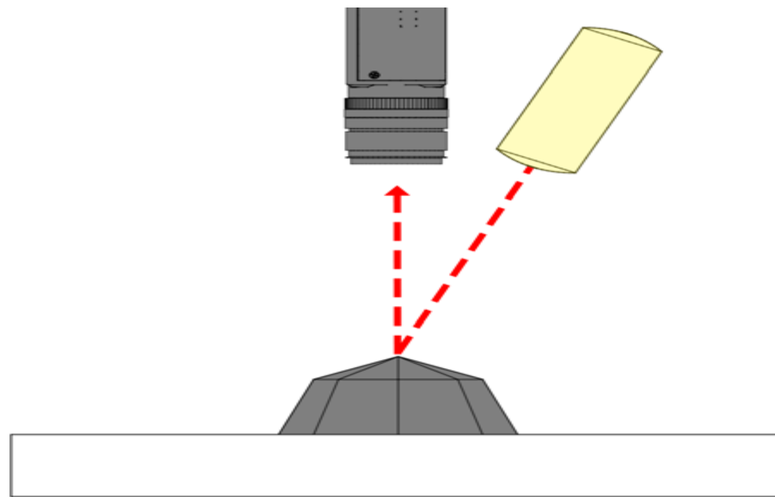
In dark-field illumination, specular light is reflected away from the camera, and diffused light from surface texture and elevation changes are reflected into the camera.



5. Bright-field illumination

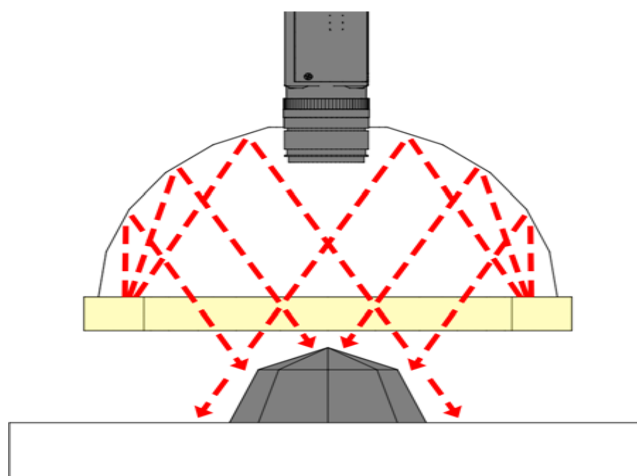
Bright-field illumination is ideal for high-contrast applications. However, highly directional light sources such as high-pressure sodium and quartz halogen may produce sharp shadows and generally do not provide consistent illumination throughout the entire field of view.

Consequently, hot-spots and specular reflections on shiny or reflective surfaces may require a more diffused light source to provide even illumination in the brightfield.



6. Diffused dome lighting

Diffused dome lighting gives the most uniform illumination of features of interest, and can mask irregularities that are not of interest and may be confusing to the scene.



7. Strobe lighting

Strobe lighting is used in high-speed applications to freeze moving objects for examination. Using a strobe light also helps to prevent blurring.

Check your Progress-4

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Lighting

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

ii. Define Back lighting

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

iii. Define Axial diffuse lighting

……………………………………………………………………………..

…..…………………………………………………………………………

……………………………………………………………………………..

iv. Define Structured light

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

v. Define Directional lighting

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

vi. Define Dark-field illumination

……………………………………………………………………………..

…..……………………………………………………………………………

……………………………………………………………………………..

vii. Define Bright-field illumination

……………………………………………………………………………..

…..……………………………………………………………………………

……………………………………………………………………………..

……………………………………………………………………………..

viii. Define Diffused dome lighting

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

ix. Define Strobe lighting

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

x. List out the advantages of lightings

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

xi. List out the disadvantages of lightings

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

12.5 Types of Machine Vision Systems

Broadly speaking, there are 3 categories of machine vision systems: 1D, 2D and 3D.

1. 1D Vision Systems

1D vision analyzes a digital signal one line at a time instead of looking at a whole picture at once, such as assessing the variance between the most recent group of ten acquired lines and an earlier group.

This technique commonly detects and classifies defects on materials manufactured in a continuous process, such as paper, metals, plastics, and other non-woven sheet or roll goods.

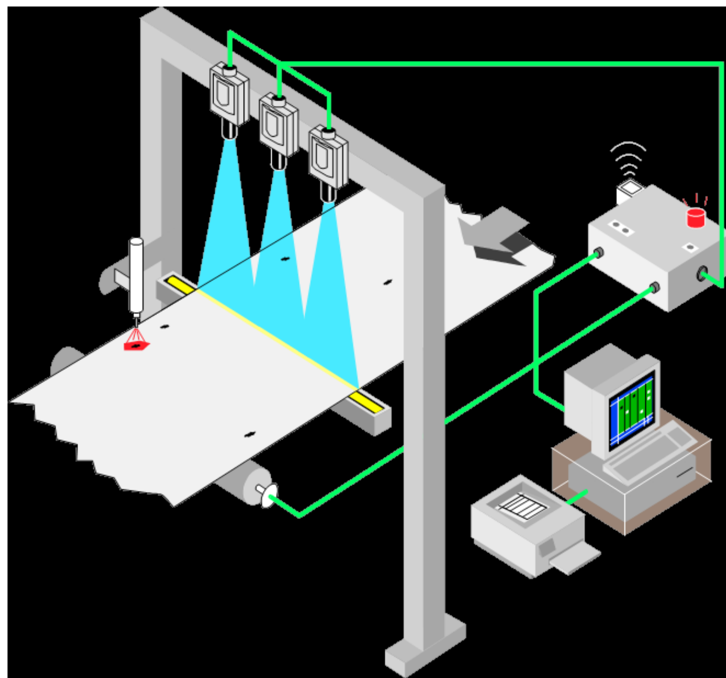


Figure 5. 1D vision systems scan one line at a time while the process moves.

2. 2D Vision Systems

Most common inspection cameras perform area scans that involve capturing 2D snapshots in various resolutions, as shown in Figure 6.

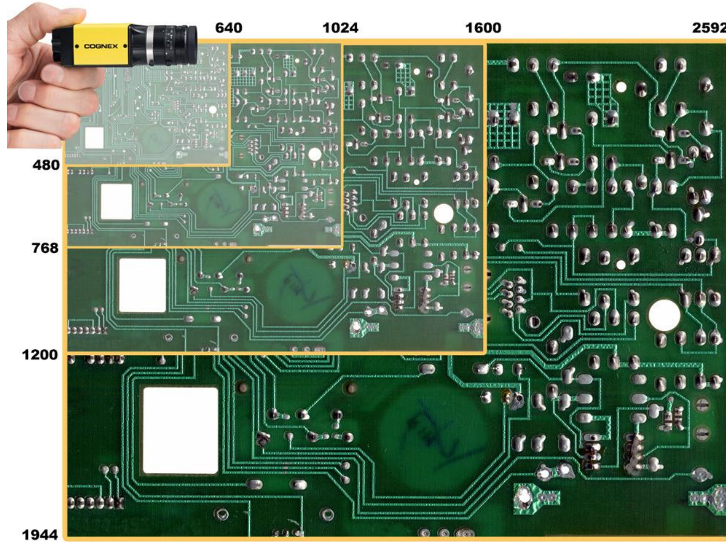


Figure 6. 2D vision systems can produce images with different resolutions.

Area Scan VS. Line Scan

In certain applications, line scan systems have specific advantages over area scan systems. For example, inspecting round or cylindrical parts may require multiple area scan cameras to cover the entire part surface.

However, rotating the part in front of a single line scan camera captures the entire surface by unwrapping the image. Line scan systems fit more easily into tight spaces for instances when the camera must peek through rollers on a conveyor to view the bottom of a part.

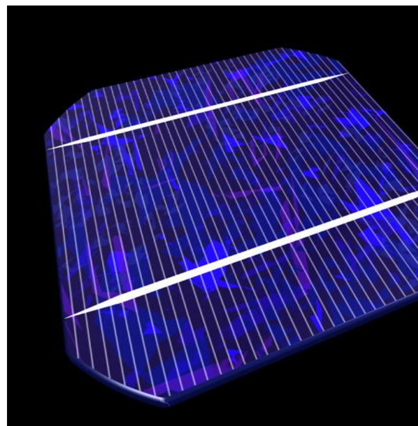
Line scan systems can also generally provide much higher resolution than traditional cameras. Since line scan systems require parts in motion to build the image, they are often well-suited for products in continuous motion.



a.



b.



c.



d.

Figure 7. Line scan cameras can (a.) unwrap cylindrical objects for inspection, (b.) add vision to space-constrained environments, (c.) meet high-resolution inspection requirements, and (d.) inspect objects in continuous motion.

3. 3D Systems

3D machine vision systems typically comprise multiple cameras or one or more laser displacement sensors. Multi-camera 3D vision in robotic guidance applications provides the robot with part orientation information.

These systems involve multiple cameras mounted at different locations and "triangulation" on an objective position in 3-D space.

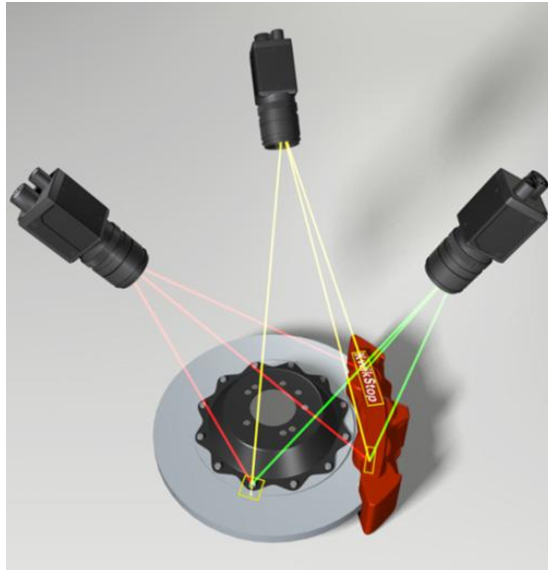


Figure 8. 3D vision systems typically employ multiple cameras.

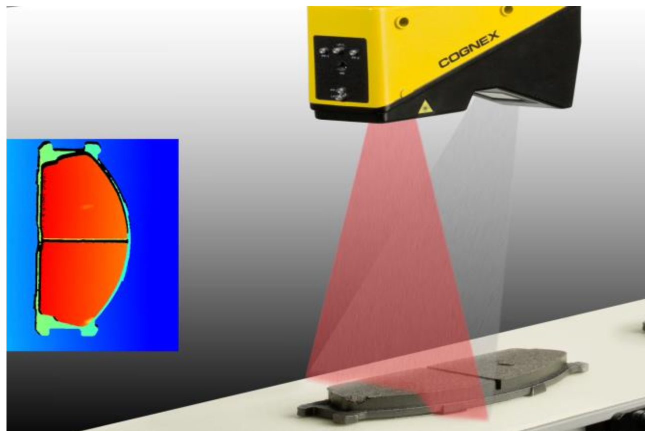


Figure 9. 3D inspection system using a single camera.

Check your Progress-5

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define 1D Vision Systems

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

ii. Define 2D Vision Systems

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

……………………………………………………………………………..

iii. Define 3D Vision Systems

……………………………………………………………………………..

…...…………………………………………………………………………

……………………………………………………………………………..

……………………………………………………………………………..

12.6 MACHINE VISION PLATFORMS

Machine vision implementation comes on several physical platforms, including PCbased systems, vision controllers designed for 3D and multi-camera 2D applications, standalone vision systems, simple vision sensors, and image-based barcode readers.

Choosing the right machine vision platform generally depends on the application's requirements, including development environment, capability, architecture, and cost.

1. PC-BASED MACHINE VISION

PC-based systems easily interface with direct-connect cameras or image acquisition boards and are well supported with configurable machine vision application software.

In addition, PCs provide a wealth of custom code development options using familiar and well-supported languages such as Visual C/C++, Visual Basic, and Java, plus graphical programming environments.

However, development tends to be long and complicated, so is usually limited to large installations and appeal mostly to advanced machine vision users and programmers.

2. VISION CONTROLLERS

Vision controllers offer all of the power and flexibility of PC-based system, but are better able to withstand the rigors of harsh factory environments.

Vision controllers allow for easier configuration of 3D and multi-camera 2D applications, perhaps for one-off tasks where a reasonable amount of time and money is available for development.

This allows for more sophisticated applications to be configured in a very cost-effective way.

3. STANDALONE VISION SYSTEMS

Standalone vision systems are cost effective and can be quickly and easily configured.

These systems come complete with the camera sensor, processor, and communications. Some also integrate lighting and autofocus optics.

In many cases these systems are compact and affordable enough to be installed throughout the factory.

By using standalone vision systems at key process points, defects can be caught earlier in the manufacturing process and equipment problems can be identified more quickly.

Most offer built-in Ethernet communications, which enables users to not only distribute vision throughout the process, but to link two or more systems together in a fully manageable, scalable vision area network in which data is exchanged between systems and managed by a host.

A network of vision systems can also be easily uplinked to plant and enterprise networks, allowing any workstation in the factory with TCP/IP capability to remotely view vision results, images, statistical data, and other information.

These systems offer configurable environments that provide easy guided setup or more advanced programming and scripting.

Some standalone vision systems provide both development environments allowing for easy set up with the added power, and flexibility of programming and scripting for greater control of system configuration and handling of vision-application data.

4. VISION SENSORS AND IMAGE-BASED BARCODE READERS

Vision sensors and image-based barcode readers generally require no programming, and provide user-friendly interfaces.

Most are easily integrated with any machine to provide single-point inspections with dedicated processing, and offer built-in Ethernet communications for factory-wide networkability.

Check your Progress-6

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define pc-based machine vision

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

ii. Define vision controllers

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

………………………………………………………………………………..

iii. Define standalone vision systems

………………………………………………………………………………..

…..……………………………………………………………………………

………………………………………………………………………………..

………………………………………………………………………………..

12.7 A-D Conversion

An analog-to-digital converter (ADC, A/D, or A-to-D) is a system that converts an analog signal, such as a sound picked up by a microphone or light entering a digital camera, into a digital signal.

An ADC may also provide an isolated measurement such as an electronic device that converts an input analog voltage or current to a digital number representing the magnitude of the voltage or current.

An ADC converts a continuous-time and continuous-amplitude analog signal to a discrete-time and discrete-amplitude digital signal.

The conversion involves quantization of the input, so it necessarily introduces a small amount of error or noise.

Furthermore, instead of continuously performing the conversion, an ADC does the conversion periodically, sampling the input, limiting the allowable bandwidth of the input signal.

The performance of an ADC is primarily characterized by its bandwidth and signal-to-noise ratio (SNR).

The bandwidth of an ADC is characterized primarily by its sampling rate. The SNR of an ADC is influenced by many factors, including the resolution, linearity and accuracy, aliasing and jitter.

The SNR of an ADC is often summarized in terms of its effective number of bits (ENOB), the number of bits of each measure it returns that are on average not noise.

An ideal ADC has an ENOB equal to its resolution. ADCs are chosen to match the bandwidth and required SNR of the signal to be digitized.

If an ADC operates at a sampling rate greater than twice the bandwidth of the signal, then per the Nyquist–Shannon sampling theorem, perfect reconstruction is possible.

The presence of quantization error limits the SNR of even an ideal ADC. However, if the SNR of the ADC exceeds that of the input signal, its effects may be neglected resulting in an essentially perfect digital representation of the analog input signal.

Check your Progress-7

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define An analog-to-digital converter

…………………………………………………………………………..

…………………………………………………………………………..

…………………………………………………………………………..

…………………………………………………………………………..

…………………………………………………………………………..

ii. Define signal-to-noise ratio

…………………………………………………………………………..

…………………………………………………………………………..

…………………………………………………………………………..

…………………………………………………………………………..

…………………………………………………………………………..

iii. Define Nyquist–Shannon sampling theorem

…………………………………………………………………………..

…..…………………………………………………………………………

…………………………………………………………………………..

…………………………………………………………………………..

12.8 Quantization

Quantization is the process of constraining an input from a continuous or otherwise large set of values (such as the real numbers) to a discrete set (such as the integers).

Quantization, in mathematics and digital signal processing, is the process of mapping input values from a large set to output values in a smaller set, often with a finite number of elements.

Rounding and truncation are typical examples of quantization processes. In this process each samplified discret-time voltage level is assigned to a finite no of definite amplitude levels.

12.9 Encoding Imaging Storage

Encoding is defined as the initial learning of information; storage refers to maintaining information over time; retrieval is the ability to access information when you need it.

12.7.1 Memory Processes

Memory is essentially the capacity for storing and retrieving information. Three processes are involved in memory: encoding, storage, and retrieval.

All three of these processes determine whether something is remembered or forgotten.

Encoding

Processing information into memory is called Encoding. People automatically encode some types of information without being aware of it.

12.7.2 Types of Encoding

There are several different ways of encoding verbal information:

- **Structural Encoding** focuses on what words look like. For instance, one might note whether words are long or short, in uppercase or lowercase, or handwritten or typed.

- **Phonemic Encoding** focuses on how words sound.

- **Semantic Encoding** focuses on the meaning of words. Semantic encoding requires a deeper level of processing than structural or phonemic encoding and usually results in better memory.

Encoding

The first process that your brain performs when it gets new information is encoding.

Compare your brain to a computer. When data is entered into a computer, it's encoded or put into a format that the computer can store.

Take digital images, for example. In the computer, an image is actually encoded in a grid of colored dots called pixels. This special format allows the computer to store images.

Storage

Once information is encoded, it can be stored. While you're still using it, the information is stored in your computer's RAM, which is used only for short-term storage.

This is like your brain's short-term memory. When you save the file, it's like putting the information in long-term memory, which is like the brain's hard drive.

The information stored in RAM will not be there after you reboot your computer. But the hard drive is like your long-term memory, and the information is there permanently.

Check your Progress-8

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Encoding

…………………………………………………………………………………..

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

ii. Define Storage

…………………………………………………………………………………..

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

iii. Define Semantic Encoding

…………………………………………………………………………………..

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

iv. Define Structural Encoding

…………………………………………………………………………………..

…………………………………………………………………………………

12.8 Image Data Reduction

It is used to reduce the volume of data. There are two types of techniques we use.

1. **Digital Conversion:**
   It is used to reduce no of gray levels used by machine vision system.

   Example:

   For a 8 bit register for each pixel there would be 2^8=256 grey levels
   For 4 bits it is 2^4=16 grey levels.

2. **Windowing:**

   It uses only a portion of the image stored in frame buffer for image processing and analysis.

   Windowing is the process of selecting some segment of the total pixel value range (the wide dynamic range of the receptors) and then displaying the pixel values within that segment over the full brightness (shades of gray) range from white to black.

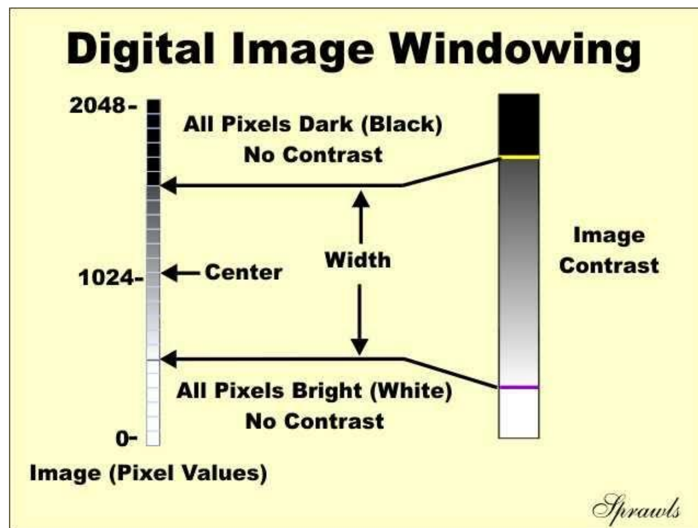


Figure 3: Digital Image Windowing

12.9 Unit – End Exercises

1. List out the imaging devices
2. Benefits of Machine Vision
3. Define Encoding
4. Define Storage
5. Define Semantic Encoding
6. Define Structural Encoding
7. Define An analog-to-digital converter
8. Define Windowing
9. Define Digital Conversion
10. Define Quantization
11. Define signal-to-noise ratio

12.10 Answer to Check your Progress

1. The camera’s ability to capture a correctly-illuminated image of the inspected object depends not only on the lens, but also on the image sensor within the camera.

    a. Input and output hardware

    b. Lenses

    c. Light sources, such as LED illuminators or halogen lamps

    d. An image processing program

    e. A sensor to detect and trigger image acquisition

    f. Actuators to sort defective parts

2. Connected Imaging Devices. Connected imaging devices include digital still cameras (DSCs) and digital camcorders with embedded wireless LAN (WLAN) or wireless WAN digital cellular connections.

3. Processing information into memory is called Encoding. People automatically encode some types of information without being aware of it.

4. Once information is encoded, it can be stored. While you're still using it, the information is stored in your computer's RAM, which is used only for short-term storage.

5. Focuses on the meaning of words. Semantic encoding requires a deeper level of processing than structural or phonemic encoding and usually results in better memory.

6. Focuses on what words look like. For instance, one might note whether words are long or short, in uppercase or lowercase, or handwritten or typed.

7. An analog-to-digital converter (ADC, A/D, or A-to-D) is a system that converts an analog signal, such as a sound picked up by a microphone or light entering a digital camera, into a digital signal.

8. Windowing is the process of selecting some segment of the total pixel value range (the wide dynamic range of the receptors) and then displaying the pixel values within that segment over the full brightness (shades of gray) range from white to black.

9. Digital Conversion
   It is used to reduce no of gray levels used by machine vision system.

   Example:

   For a 8 bit register for each pixel there would be 2^8=256 grey levels
   For 4 bits it is 2^4=16 grey levels.

10. Quantization is the process of constraining an input from a continuous or otherwise large set of values (such as the real numbers) to a discrete set (such as the integers).

11. The SNR of an ADC is often summarized in terms of its effective number of bits (ENOB), the number of bits of each measure it returns that are on average not noise.

12.11 Suggested Readings

1. E. R. Davies (2004), Morgan Kaufmann, Machine Vision 3rd Edition Theory, Algorithms, Practicalities, UK.

2. Ramesh Jain, Rangachar Kasturi, Brian G. Schunck (1995), Published by McGraw-Hill, Inc., MACHINE VISION, ISBN 0-07-032018-7.

3. http://www.roboticsbible.com/machine-vision-system.html

4. http://what-when-how.com/metrology/how-machine-vision-system-functions-metrology/

5. Milan Sonka Vaclav Hlavac Roger Boyle (2001), PWS Publishing Company, Second Edition, Image Processing, Analysis, and Machine Vision.

NOTES

UNIT - XIII SEGMENTATION TECHNIQUES

Structure

13.1 Introduction

Segmentation is the process of breaking an image into groups, based on similarities of the pixels.

The basic idea is the following: Each image pixel can be associated with certain visual properties, such as brightness, color, and texture.

Within an object, or a single part of an object, these attributes vary relatively little, whereas across an inter-object boundary there is typically a large change in one or the other of these attributes.

We need to find a partition of the image into sets of pixels such that these constraints are satisfied as well as possible.

Segmentation based purely on low-level, local attributes, such as brightness and color is an error-prone process.

To reliably find boundaries associated with objects, one should also incorporate high-level knowledge of the kinds of objects one may expect to encounter in a scene.

Segmented the images have been segmented to separate objects from the background.

Image segmentation is a method which can be used to understand images and extract information or objects.

It is the first step in image analysis. Feature extraction in image processing is a technique of redefining a large set of redundant data into a set of features (or feature vector) of reduced dimension.

Feature extraction a type of dimensionality reduction that efficiently represents interesting parts of an image as a compact feature vector.

This approach is useful when image sizes are large and a reduced feature representation is required to quickly complete tasks such as image matching and retrieval.

Check your Progress-1

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Segmentation

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

13.2 Feature Extraction

Image features, such as edges and interest points, provide rich information on the image content.

They correspond to local regions in the image and are fundamental in many applications in image analysis: recognition, matching, reconstruction, etc.

Image features yield two different types of problem: the detection of area of interest in the image, typically contours, and the description of local regions in the image, typically for matching in different images.

In any case, they relate to the differential properties of the intensity function, for instance the gradient or the laplacian that are used to detect intensity discontinuities that occur at contours.

The feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations.

Feature extraction is related to dimensionality reduction. When the input data to an algorithm is too large to be processed and it is suspected to be redundant (e.g. the same measurement in both feet and meters, or the repetitiveness of images presented as pixels), then it can be transformed into a reduced set of features (also named a feature vector).

Determining a subset of the initial features is called feature selection. The selected features are expected to contain the relevant information from the input data, so that the desired task can be performed by using this reduced representation instead of the complete initial data.

The thresholding is a form of low-level feature extraction performed as a point operation. All of these approaches can be used in high-level feature extraction, where we find shapes in images.

It is well known that we can recognize people from caricaturists' portraits. That is the first low-level feature we shall encounter. It is called edge detection.

Low-level features to be those basic features that can be extracted automatically from an image without any shape information (information about spatial relationships).

A feature extraction step reduces the data by measuring certain properties or features of the labeled objects.

These features (or, more precisely, the values of these features) are then passed to a classifier that evaluates the evidence presented and makes a decision as to the class each object should be assigned.

13.2.1 Types of Low-level Features

- **Edge detection**

  Edge detection includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities.

  The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges.

- **Corner detection**

  Corner detection is an approach used within computer vision systems to extract certain kinds of features and infer the contents of an image.

- **Blob detection**

  Blob detection methods are aimed at detecting regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions.

Informally, a blob is a region of an image in which some properties are constant or approximately constant; all the points in a blob can be considered in some sense to be similar to each other. The most common method for blob detection is convolution.

- **Ridge detection**

  The ridges (or the ridge set) of a smooth function of two variables are a set of curves whose points are, in one or more ways to be made precise below, local maxima of the function in at least one dimension. This notion captures the intuition of geographical ridges.

- **Scale-Invariant Feature Transform**

  SIFT key points of objects are first extracted from a set of reference images and stored in a database.

  An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors.

13.2.2 Shape based Features

- **Thresholding**

  The simplest thresholding methods replace each pixel in an image with a black pixel if the image intensity $I_{i,j}$ is less than some fixed constant T (that is, $I_{i,j} < T$), or a white pixel if the image intensity is greater than that constant.

- **Template matching**

  Template matching is a technique in digital image processing for finding small parts of an image which match a template image.

  The main challenges in the template matching task are: occlusion, detection of non-rigid transformations, illumination and background changes, background clutter and scale changes.

- **Hough transform**

  The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing.

  The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure.

  The classical Hough transform was concerned with the identification of lines in the image, but later the Hough transform has been extended to identifying positions of arbitrary shapes, most commonly circles or ellipses.

13.2.3 Feature Detection

Many types of features are used for object recognition. Most features are based on either regions or boundaries in an image.

It is assumed that a region or a closed boundary corresponds to an entity that is either an object or a part of an object. Some of the commonly used features are as follows.

- **Global Features**

Global features usually are some characteristics of regions in images such as area (size), perimeter, Fourier descriptors, and moments.

Global features can be obtained either for a region by considering all points within a region, or only for those points on the boundary of a region.

In each case, the intent is to find descriptors that are obtained by considering all points, their locations, intensity characteristics, and spatial relations.

- **Local Features**

Local features are usually on the boundary of an object or represent a distinguishable small area of a region.

Curvature and related properties are commonly used as local features.

The curvature may be the curvature on a boundary or may be computed on a surface.

The surface may be an intensity surface or a surface in 2.5-dimensional space.

High curvature points are commonly called corners and play an important role in object recognition.

Local features can contain a specific shape of a small boundary segment or a surface patch. Some commonly used local features are *curvature, boundary segments,* and *corners.*

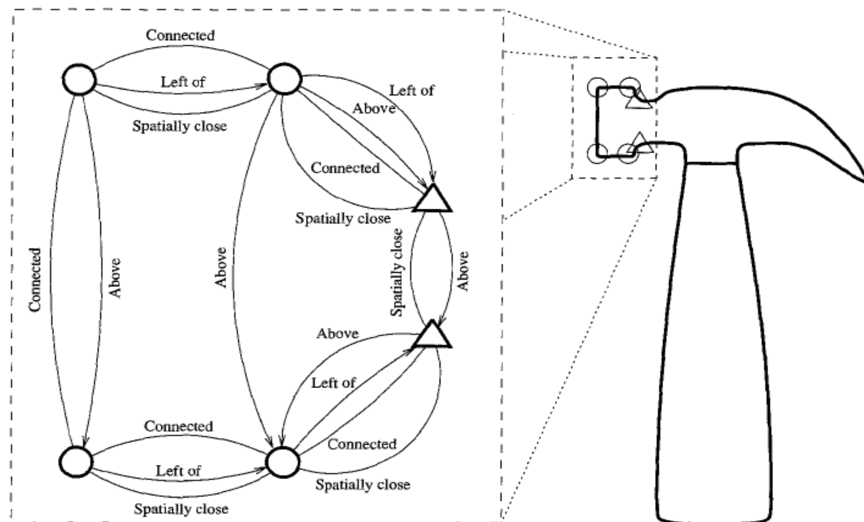
- **Relational Features**

Relational features are based on the relative positions of different entities, either regions, closed contours, or local features.

These features usually include distance between features and relative orientation measurements.

These features are very useful in defining composite objects using many regions or local features in images.

In most cases, the relative position of entities is what defines objects. The exact same feature, in slightly different relationships, may represent entirely different objects.



Number of Regions	2
Area	42574 pixels
Perimeter	1343 pixels

Figure 1. An object and its partial representation using multiple local and global features.

13.2.4 Feature Matching

Suppose that each object class is represented by its features. As above, let us assume that the j^{th} feature's value for the i^{th} class is denoted by f_{ij}. For an unknown object the features are denoted by u_j. The similarity of the object with the i^{th} class is given by

$$S_i = \sum_{j=1}^{N} w_j s_j \quad (1)$$

Where w_j is the weight for the jth feature. The weight is selected based on the relative importance of the feature. The similarity value of the jth feature is s_j. This could be the absolute difference, normalized difference, or any other distance measure. The most common method is to use

$$s_j = |u_j - f_{ij}| \quad (2)$$

and to account for normalization in the weight used with the feature.

The object is labeled as belonging to class k if s_k is the highest similarity value. Note that in this approach, we use features that may be local or global.

13.2.5 Feature Indexing

If the number of objects is very large and the problem cannot be solved using feature space partitioning, then indexing techniques become attractive.

This sequential nature of the approach makes it unsuitable with a number of objects.

In such a case, one should be able to use a hypothesizer that reduces the search space significantly.

The next step is to compare the models of each object in the reduced set with the image to recognize the object.

Feature indexing approaches use features of objects to structure the modelbase.

When a feature from the indexing set is detected in an image, this feature is used to reduce the search space.

More than one feature from the indexing set may be detected and used to reduce the search space and in turn reduce the total time spent on object recognition.

The features in the indexing set must be determined using the knowledge of the modelbase.

If such knowledge is not available, a learning scheme should be used. This scheme will analyze the frequency of each feature from the feature set and, based on the frequency of features, form the indexing set, which will be used for structuring the database.

In the indexed database, in addition to the names of the objects and their models, information about the orientation and pose of the object in which the indexing feature appears should always be kept.

This information helps in the verification stage. Once the candidate object set has been formed, the verification phase should be used for selecting the best object candidate.

Check your Progress-2

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. List out Types of Low-level Features

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

ii. Define Edge detection

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

iii. Define Corner detection

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

iv. Define Blob detection

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

v. Define Ridge detection

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

vi. Define Scale-Invariant Feature Transform

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

13. 3 Object Recognitions

Object recognition is the ability to perceive an object's physical properties (such as shape, color and texture) and apply semantic attributes to it (such as identifying the object as an apple).

This process includes the understanding of its use, previous experience with the object, and how it relates to others.

- **Biometric identification**

  Criminal investigations and access control for restricted facilities require the ability to identify unique individuals.

  - Fingerprints
  - Iris scans
  - Facial photographs

  Result in images that must be matched to specific individuals.

- **Content-based image retrieval**

  It is easy to find a location in a document, if one exists, for the string "cat'-any text editor provides this capability.

  Now consider the problem of finding the subset of pixels in an image which correspond to the image of a cat.

The use of shape for object recognition has proved to be much more difficult.

There are two main approaches:

- Brightness-based recognition: in which pixel brightness values are used directly.
- Feature-based recognition: which involves the use of spatial arrangements of extracted features such as edges or key points.

13.3.1 Brightness-based recognition

The subset of image pixels that corresponds to a candidate object, define the features to be the raw pixel brightness values themselves.

One negative aspect of using raw pixels as feature vectors is the great redundancy inherent in this representation.

Consider two nearby pixels the cheek of a face; they are likely to be very highly correlated because of similar geometry, illumination, etc.

Data reduction techniques, such as principal component analysis, can be used successfully to reduce the dimensionality of the feature vector, enabling recognition of such things as faces with greater speed than one would get in a higher-dimensional space.

13.3.2 Feature-based recognition

The raw pixel brightnesses as features, we can detect and mark spatially localized features such as regions and edges.

There are two motivations for using edges. One is data reduction-there are far fewer edges than image pixels.

The other is illumination invariance-within a suitable range of contrasts, the edges will be detected at roughly the same locations, independent of precise lighting configuration.

Edges are one-dimensional features; two-dimensional features (regions) and zero-dimensional features (points) have also been used.

Many different definitions have been proposed for distances between images.

One of the more interesting approaches is based on the idea of deformable matching.

A notion of shape similarity as a three stage process:

(1) Solve the correspondence problem between the two shapes

(2) Use the correspondences to estimate an aligning transform

(3) Compute the 'distance between the two shapes as a sum of matching errors between corresponding points, together with a term measuring the magnitude of the aligning transformation.

The shape context, which describes the coarse arrangement of the rest of the shape with respect to the point.

Object Recognition

An object recognition system finds objects in the real world from an image of the world, using object models which are known a priori. This task is surprisingly difficult.

Humans perform object recognition effortlessly and instantaneously. Algorithmic description of this task for implementation on machines has been very difficult.

The object recognition problem can be defined as a labeling problem based on models of known objects.

Formally, given an image containing one or more objects of interest (and background) and a set of labels corresponding to a set of models known to the system, the system should assign correct labels to regions, or a set of regions, in the image.

The object recognition problem is closely tied to the segmentation problem: without at least a partial recognition of objects, segmentation cannot be done, and without segmentation, object recognition is not possible.

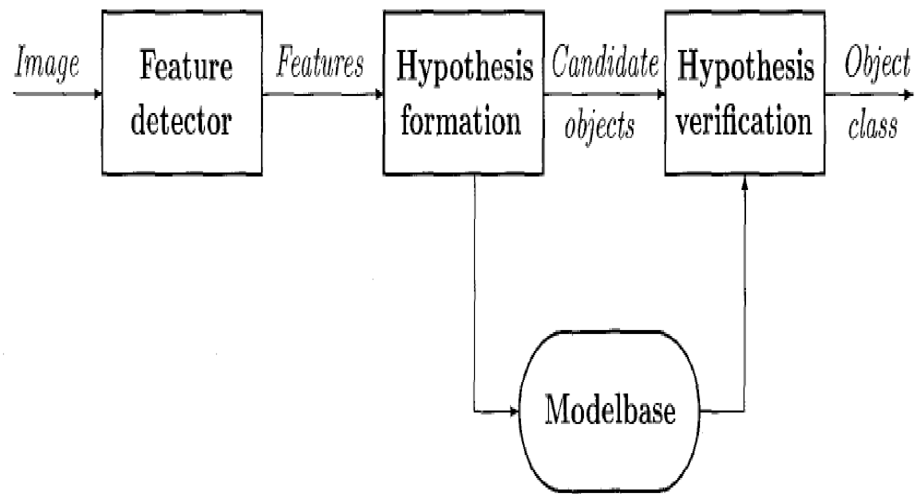


Figure 2. Different components of an object recognition system

13.3.3 System Componens

An object recognition system must have the following components to perform the task:

- Model database (also called modelbase)
- Feature detector
- Hypothesizer
- Hypothesis verifier

The model database contains all the models known to the system. The information in the model database depends on the approach used for the recognition.

It can vary from a qualitative or functional description to precise geometric surface information. A feature is some attribute of the object that is considered important in describing and recognizing the object in relation to other objects. Size, color, and shape are some commonly used features.

The feature detector applies operators to images and identifies locations of features that help in forming object hypotheses. The features used by a system depend on the types of objects to be recognized and the organization of the model database.

Using the detected features in the image, the hypothesizer assigns likelihoods to objects present in the scene. This step is used to reduce the search space for the recognizer using certain features.

The modelbase is organized using some type of indexing scheme to facilitate elimination of unlikely object candidates from possible consideration.

The verifier then uses object models to verify the hypotheses and refines the likelihood of objects. The system then selects the object with the highest likelihood, based on all the evidence, as the correct object.

All object recognition systems use models either explicitly or implicitly and employ feature detectors based on these object models. The hypothesis formation and verification components vary in their importance in different approaches to object recognition.

Some systems use only hypothesis formation and then select the object with highest likelihood as the correct object. Pattern classification approaches are a good example of this approach.

Many artificial intelligence systems, on the other hand, rely little on the hypothesis formation and do more work in the verification phases. In fact, one of the classical approaches, template matching, bypasses the hypothesis formation stage entirely.

An object recognition system must select appropriate tools and techniques for the steps discussed above. Many factors must be considered in the selection of appropriate methods for a particular application.

The central issues that should be considered in designing an object recognition system are:

- **Object or model representation**

The representation of an object should capture all relevant information without any redundancies and should organize this information in a form that allows easy access by different components of the object recognition system.

- **Feature extraction**

Most features can be computed in two dimensional images but they are related to three-dimensional characteristics of objects. Due to the nature of the image formation process, some features are easy to compute reliably while others are very difficult.

- **Feature-model matching**

An exhaustive matching approach will solve the recognition problem but may be too slow to be useful. Effectiveness of features and efficiency of a matching technique must be considered in developing a matching approach.

- **Hypotheses formation**

The hypothesis formation step is basically a heuristic to reduce the size of the search space. This step uses knowledge of the application domain to assign some kind of probability or confidence measure to different objects in the domain.

This measure reflects the likelihood of the presence of objects based on the detected features.

- **Object verification**

The presence of each likely object can be verified by using their models. One must examine each plausible hypothesis to verify the presence of the object or ignore it.

If the models are geometric, it is easy to precisely verify objects using camera location and other scene parameters. In other cases, it may not be possible to verify a hypothesis.

13.3.4 Complexity of Object Recognition

A qualitative way to consider the complexity of the object recognition task would consider the following factors:

- **Scene constancy**

The scene complexity will depend on whether the images are acquired in similar conditions (illumination, background, camera parameters, and viewpoint).

Under different scene conditions, the performance of different feature detectors will be significantly different.

The nature of the background, other objects, and illumination must be considered to determine what kind of features can be efficiently and reliably detected.

- **Image-models spaces**

In some applications, images may be obtained such that three-dimensional objects can be considered two-dimensional. The models in such cases can be represented using two-dimensional characteristics.

If models are three-dimensional and perspective effects cannot be ignored, then the situation becomes more complex.

In this case, the features are detected in two-dimensional image space, while the models of objects may be in three-dimensional space.

Thus, the same three-dimensional feature may appear as a different feature in an image. This may also happen in dynamic images due to the motion of objects.

- **Number of objects in the model database**

If the number of objects is very small, one may not need the hypothesis formation stage.

A sequential exhaustive matching may be acceptable. Hypothesis formation becomes important for a large number of objects.

The amount of effort spent in selecting appropriate features for object recognition also increases rapidly with an increase in the number of objects.

- **Number of objects in an image and possibility of occlusion**

If there is only one object in an image, it may be completely visible. With an increase in the number of objects in the image, the probability of occlusion increases.

Occlusion is a serious problem in many basic image computations. Occlusion results in the absence of expected features and the generation of unexpected features.

Occlusion should also be considered in the hypothesis verification stage. Generally, the difficulty in the recognition task increases with the number of objects in an image.

Difficulties in image segmentation are due to the presence of multiple occluding objects in images.

- **Two-dimensional**

In many applications, images are acquired from a distance sufficient to consider the projection to be orthographic.

If the objects are always in one stable position in the scene, then they can be considered two-dimensional.

In these applications, one can use a two-dimensional modelbase. There are two possible cases:

1. Objects will not be occluded, as in remote sensing and many industrial applications.

2. Objects may be occluded by other objects of interest or be partially visible, as in the bin of parts problem.

In some cases, though the objects may be far away, they may appear in different positions resulting in multiple stable views.

- **Three-dimensional**

If the images of objects can be obtained from arbitrary viewpoints, then an object may appear very different in its two views.

For object recognition using three-dimensional models, the perspective effect and viewpoint of the image have to be considered.

The fact that the models are three-dimensional and the images contain only two-dimensional information affects object recognition approaches.

Again, the two factors to be considered are whether objects are separated from other objects or not. For three-dimensional cases, one should consider the information used in the object recognition task.

Two different cases:

1. **Intensity**

   There is no surface information available explicitly in intensity images. Using intensity values, features corresponding to the three-dimensional structure of objects should be recognized.

2. **2.5-dimensional images**

   In many applications, surface representations with viewer-centered coordinates are available, or can be computed, from images.

   This information can be used in object recognition. Range images are also 2.5-dimensional. These images give the distance to different points in an image from a particular view point.

13.3.5 Object Representation

Images represent a scene from a camera's perspective. It appears natural to represent objects in a camera-centric, or viewer-centered, coordinate system. Another possibility is to represent objects in an object-centered coordinate system.

Since it is easy to transform from one coordinate system to another using their relative positions, the central issue in selecting the proper coordinate system to represent objects is the ease of representation to allow the most efficient representation for feature detection and subsequent processes.

A representation allows certain operations to be efficient at the cost of other operations. Representations for object recognition are no exception. Designers must consider the parameters in their design problems to select the best representation for the task.

1. **Observer-Centered Representations**

If objects usually appear in a relatively few stable positions with respect to the camera, then they can be represented efficiently in an observer-centered coordinate system.

If a camera is located at a fixed position and objects move such that they present only some aspects to the camera, then one can represent objects based on only those views.

If the camera is far away from objects, as in remote sensing, then three-dimensionality of objects can be ignored. In such cases, the objects can be represented only by a limited set of views-in fact, only one view in most cases.

Finally, if the objects in a domain of applications are significantly different from each other, then observer-centered representations may be enough.

Observer-centered representations are defined in image space. These representations capture characteristics and details of the images of objects in their relative camera positions.

One of the earliest and most rigorous approaches for object recognition is based on characterizing objects using a feature vector. This feature vector captures essential characteristics that help in distinguishing objects in a domain of application.

The features selected in this approach are usually global features of the images of objects. These features are selected either based on the experience of a designer or by analyzing the efficacy of a feature in grouping together objects of the same class while discriminating it from the members of other classes.

Many feature selection techniques have been developed in pattern classification. These techniques study the probabilistic distribution of features of known objects from different classes and use these distributions to determine whether a feature has sufficient discrimination power for classification.

An object is represented as a point in this space. It is possible that different features have different importance and that their units are different. These problems are usually solved by assigning different weights to the features and by normalizing the features.

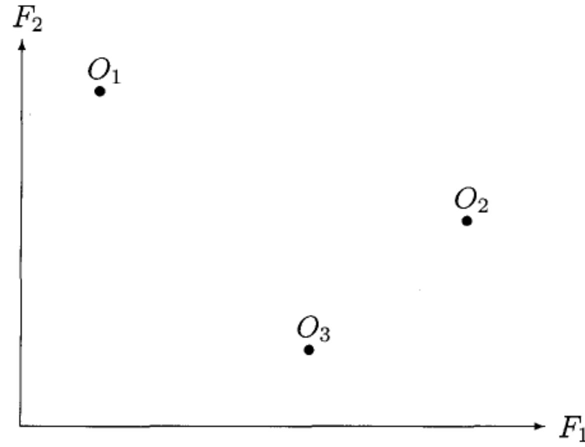


Figure 3. Two-dimensional feature space for object recognition.

2. Object-Centered Representations

An object-centered representation uses description of objects in a coordinate system attached to objects. This description is usually based on three dimensional features or description of objects.

Object-centered representations are independent of the camera parameters and location. Thus, to make them useful for object recognition, the representation should have enough information to produce object images or object features in images for a known camera and viewpoint. This requirement suggests that object-centered representations should capture aspects of the geometry of objects explicitly.

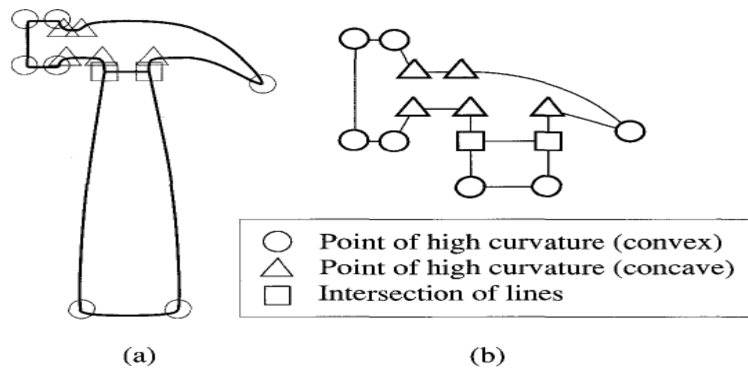


Figure 4. In (a) an object is shown with its prominent local features highlighted. A graph representation of the object is shown in (b). This representation is used for object recognition using a graph matching approach.

- **Constructive Solid Geometry**

A CSG representation of an object uses simple volumetric primitives, such as blocks, cones, cylinders, and spheres.

A set of Boolean operations: union, intersection, and difference. Since arbitrarily curved objects cannot be represented using just a few chosen primitives, CSG approaches are not very useful in object recognition.

- **Spatial Occupancy**

An object in three-dimensional space may be represented by using non overlapping sub regions of the three-dimensional space occupied by an object.

In addition to simple occupancy, one may consider representing other properties of objects at points in space. There are many variants of this representation such as voxel representation, octree, and tetrahedral cell decomposition.

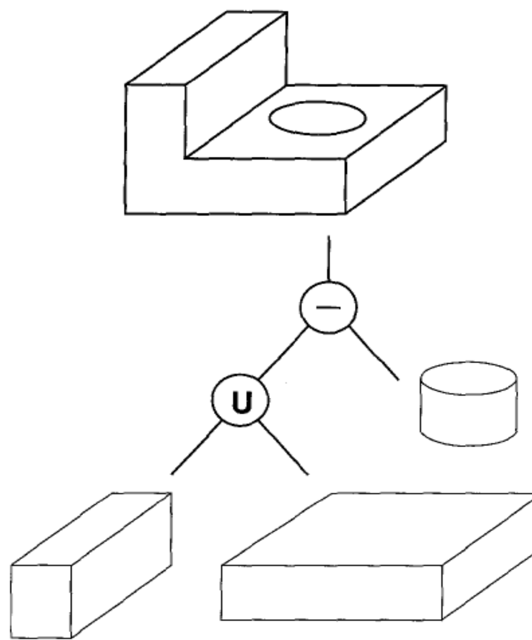


Figure 5. A CSG representation of an object uses some basic primitives and operations among them to represent an object.

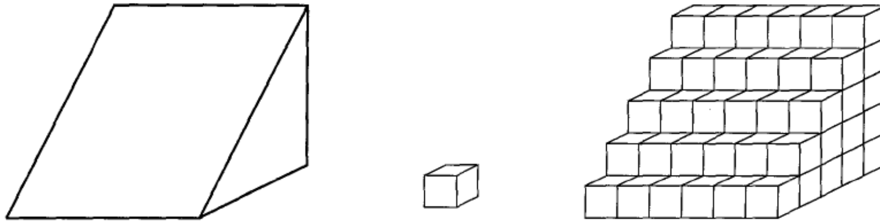


Figure 6. A voxel representation of an object.

- **Multiple-View Representation**

Since objects must be recognized from images, one may represent a three dimensional object using several views obtained either from regularly spaced viewpoints in space or from some strategically selected viewpoints.

For a limited set of objects, one may consider arbitrarily many views of the object and then represent each view in an observer-centered representation.

A three-dimensional object can be represented using its aspect graph. An aspect graph represents all stable views of an object. Thus, an aspect graph is obtained by partitioning the view-space into areas in which the object has stable views.

- **Surface-Boundary Representation**

A solid object can be represented by defining the surfaces that bound the object. The bounding surfaces can be represented using one of several methods popular in computer graphics. These representations vary from triangular patches to nonuniform rational B-splines (NURBS).

- **Sweep Representations: Generalized Cylinders**

Object shapes can be represented by a three-dimensional space curve that acts as the spine or axis of the cylinder, a two-dimensional cross-sectional figure, and a sweeping rule that defines how the cross section is to be swept along the space curve. The cross section can vary smoothly along the axis.

For many industrial and other objects, the cross section of objects varies smoothly along an axis in space, and in such cases this representation is satisfactory. For arbitrarily shaped objects, this condition is usually not satisfied, making this representation unsuitable.

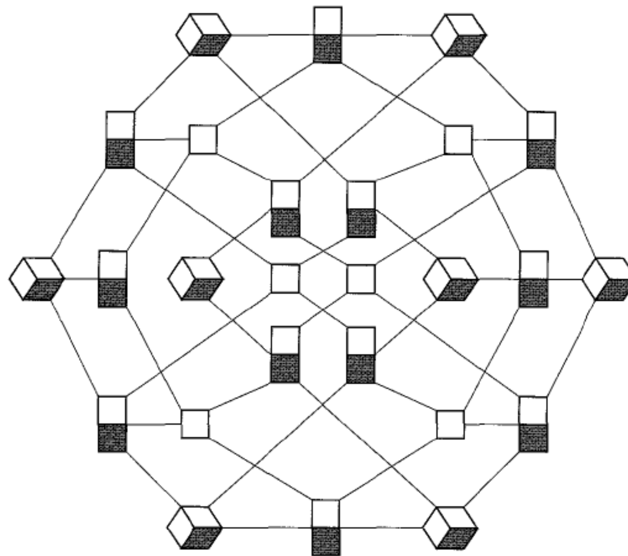


Figure 7. An object and its aspect graph. Each node in the aspect graph represents a stable view. The branches show how one can go from one stable view to other stable views through accidental views.

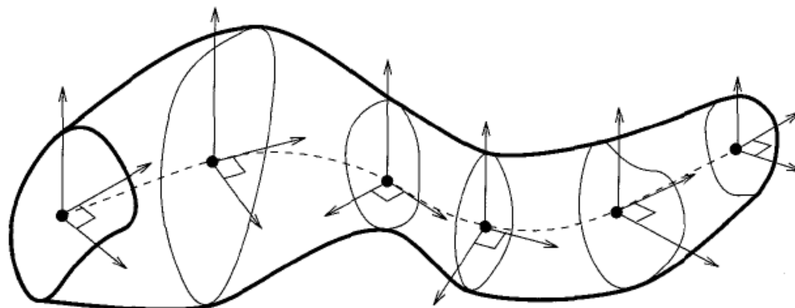


Figure 8. An object and its generalized cylinder representation. Note the axis of the cylinder is shown as a dashed line, the coordinate axes are drawn with respect to the cylinder's central axis, and the cross sections at each point are orthogonal to the cylinder's central axis.

Check your Progress-3

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Object Recognition

………………………………………………………………………………

………………………………………………………………………………

………………………………………………………………………………

………………………………………………………………………………

ii. List out System Components

………………………………………………………………………………

………………………………………………………………………………

………………………………………………………………………………

………………………………………………………………………………

iii. Define Intensity

………………………………………………………………………………

………………………………………………………………………………

………………………………………………………………………………

………………………………………………………………………………

13.4 Unit – End Exercises

1. List out Types of Low-level Features
2. Define Biometric identification
3. Object or model representation
4. Feature extraction
5. Feature-model matching
6. Hypotheses formation
7. Object verification

13.5 Answer to Check your Progress

1. Low-level features to be those basic features that can be extracted automatically from an image without any shape information (information about spatial relationships).

    a. Edge detection

    b. Corner detection

    c. Blob detection

    d. Ridge detection

    e. Scale-Invariant Feature Transform

2. Criminal investigations and access control for restricted facilities require the ability to identify unique individuals. Fingerprints, iris scans, and facial photographs result in images that must be matched to specific individuals.

3. The representation of an object should capture all relevant information without any redundancies and should organize this information in a form that allows easy access by different components of the object recognition system.

4. Most features can be computed in two dimensional images but they are related to three-dimensional characteristics of objects. Due to the nature of the image formation process, some features are easy to compute reliably while others are very difficult.

5. An exhaustive matching approach will solve the recognition problem but may be too slow to be useful. Effectiveness of features and efficiency of a matching technique must be considered in developing a matching approach.

6. The hypothesis formation step is basically a heuristic to reduce the size of the search space. This step uses knowledge of the application domain to assign some kind of probability or confidence measure to different objects in the domain.

7. The presence of each likely object can be verified by using their models. One must examine each plausible hypothesis to verify the presence of the object or ignore it. If the models are geometric, it is easy to precisely verify objects using camera location and other scene parameters. In other cases, it may not be possible to verify a hypothesis.

13.6. Suggested Readings

1. E. R. Davies (2004), Morgan Kaufmann, Machine Vision 3rd Edition Theory, Algorithms, Practicalities, UK.

2. Ramesh Jain, Rangachar Kasturi, Brian G. Schunck (1995), Published by McGraw-Hill, Inc., MACHINE VISION, ISBN 0-07-032018-7.

3. Stuart Russell and Peter Norvig (2003), Artificial Intelligence A Modern Approach, Second Edition Prentice Hall Series.

4. Mark S. Nixon, Alberto S. Aguado (2008), Feature Extraction and Image Processing, Academic Press is an imprint of Elsevier, Second edition, UK.

NOTES

UNIT - XIV TRAINING THE VISION SYSTEM

Structure

14.1 Introduction

Robots are physical agents that perform tasks by manipulating the physical world. Effectors have a single purpose: to assert physical forces on the environment.

Robots are also equipped with sensors, which allow them to perceive their environment. Manipulators, or robot arms, are physically anchored to their workplace.

Mobile robots move about their environment using wheels, legs, or similar mechanisms.

Check your Progress-1

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Robots

…………………………………………………………………………………

…………………………………………………………………………………

14.2 Robot Hardware

- **Sensors**

Sensors are the perceptual interface between robots and their environments.

1. **Passive sensors:** such as cameras, true observers of the environment.

2. **Active sensors:** such as sonar, send energy into the environment.

Active sensors tend to provide more information than passive sensors, but at the expense of increased power consumption and with a danger of interference when multiple active sensors are used at the same time.

Whether active or passive, sensors can be divided into three types, depending on whether they record distances to objects, entire images of the environment, or properties of the robot itself.

Some range sensors measure very short or very long distances. Close-range sensors include tactile sensors such as whiskers, bump panels, and touch-sensitive skin.

At the other end of the spectrum is the Global Positioning System (GPS), which measures the distance to satellites that emit pulsed signals.

Other important aspects of robot state are measured by force and torque sensors. These are indispensable when robots handle fragile objects or objects whose exact shape and location is unknown.

Dynamically stable, meaning that it can remain upright while hopping around. A robot that can remain upright without moving its legs is called statically stable.

A robot is statically stable if its center of gravity is above the polygon spanned by its legs.

Sensors and effectors alone do not make a robot. A complete robot also needs a source of power to drive its effectors.

The electric motor is the most popular mechanism for both manipulator actuation and locomotion, but pneumatic actuation using compressed gas and hydraulic actuation using pressurized fluids also have their application niches.

Most robots also have some means of digital communication such as a wireless network.

Check your Progress-2

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Sensors

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

ii. Define Passive sensors

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

iii. Define Active sensors

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

14.3 Robotic Perception

Perception is the process by which robots map sensor measurements into internal representations of the environment.

Perception is difficult because in general the sensors are noisy, and the environment is partially observable, unpredictable, and often dynamic.

- **Localization**

Localization is a generic example of robot perception. It is the problem of determining where things are.

Localization is one of the most pervasive perception problems in robotics, because knowledge about where things are is at the core of any successful physical interaction.

Check your Progress-3

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Robotic Perception

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

iv. Define Localization

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

14.4 Planning to Move

The point-to-point motion problem is to deliver the robot or its end-effector to a designated target location.

The configuration space, the space of robot states defined by location, orientation, and joint angles is a better place to work than the original 3D space.

The path planning problem is to find a path from one configuration to another in configuration space.

- **Cell decomposition methods**

The path planning are uses cell decomposition. It decomposes the free space into a finite number of contiguous regions, called cells.

- **Skeletonization methods**

The second major family of path-planning algorithms is based on the idea of skeletonization.

These algorithms reduce the robot's free space to a one-dimensional representation, for which the planning problem is easier.

This lower-dimensional representation is called a skeleton of the configuration space.

Check your Progress-4

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

ii. Define Cell decomposition method

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

ii. Define Planning to Move

………………………………………………………………………………

………………………………………………………………………………

………………………………………………………………………………

iii. Define Skeletonization methods

………………………………………………………………………………

………………………………………………………………………………

………………………………………………………………………………

14.5 Robotic applications of machine vision

- **Industry and Agriculture**

Traditionally, robots have been fielded in areas that require difficult human labor, yet are structured enough to be amenable to robotic automation.

The best example is the assembly line, where manipulators routinely perform tasks such as assembly, part placement, material handling, welding, and painting.

In many of these tasks, robots have become more cost-effective than human workers.

Outdoors, many of the heavy machines that we use to harvest, mine, or excavate earth have been turned into robots.

- **Transportation**

Robotic transportation has many facets: from autonomous helicopters that deliver objects to locations that would be hard to access by other means, to automatic wheelchairs that transport people who are unable to control wheelchairs by themselves, to autonomous straddle carriers that outperform skilled human drivers when transporting containers from ships to trucks on loading docks.

- **Hazardous environments**

Robots have assisted people in cleaning up nuclear waste, most notably in Chernobyl and Three Mile Island.

Robots were present after the collapse of the World Trade Center, where they entered structures deemed too dangerous for human search and rescue crews.

Some countries have used robots to transport ammunition and to defuse bombs a notoriously dangerous task.

- **Exploration**

Robots have gone where no-one has gone before, including the surface of Mars.

Robotic arms assist astronauts in deploying and retrieving satellites and in building the International Space Station.

Robots also help explore under the sea. Unmanned air vehicles known as drones are used in military operations.

Robots are becoming very effective tools for gathering information in domains that are difficult (or dangerous) to access for people.

- **Health care**

Robots are increasingly used to assist surgeons with instrument placement when operating on organs as intricate as brains, eyes, and hearts.

- **Personal Services**

Service is an up-and-coming application domain of robotics. Service robot assist individuals in performing daily tasks.

Commercially available domestic service robots include autonomous vacuum cleaners, lawn mowers, and golf caddies.

All these robots can navigate autonomously and perform their tasks without human help.

Some service robots operate in public places, such as robotic information kiosks that have been deployed in shopping malls and trade fairs, or in museums as tour-guides.

Service tasks require human interaction, and the ability to cope robustly with unpredictable and dynamic environments.

- **Entertainment**

Robots have begun to conquer the entertainment and toy industry.

- **Human augmentation**

A final application domain of robotic technology is that of human augmentation. Researchers have developed legged walking machines that can carry people around, very much like a wheelchair.

Several research efforts presently focus on the development devices that make it easier for people to walk or move their arms, by providing additional forces through extra-skeletal attachments.

Check your Progress-5

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Industry and Agriculture

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

ii. Define Transportation

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

iii. Define Hazardous environments

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

iv. Define Exploration

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

v. Define Health care

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

vi. Define Personal Services

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

vii. Define Entertainment

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

viii. Define Human augmentation

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

……………………………………………………………………………

14.6 Process of Image Generation in Robots

14.6.1 Process of Image Generation in Robots

1. Introduction

The vision system enables a robot to see and identify various objects. The system provides the robot with an ability to collect the light reflected from the objects, process an image out of it and then perform the desired action.

The vision system helps with these functions by using various electronic devices and hardware systems.

2. How Does the System Works?

A vision system in a robot identifies an object by forming an electronic image using a bunch of pixels already stored in the memory of the robot's controlling unit.

Each pixel has a binary number allotted to it. Each of these binary numbers represents a particular wavelength and intensity in the light spectrum.

An electronic image is formed in the controlling unit of the robot by assembling various binary numbers according to the amount of light.

3. Types of Vision Systems

A robot's vision system is classified into three main types on the basis of the color of the objects. They are:

- Binary image, which consist of black and white images
- Gray colored images
- Colored images with the base of red, green or blue

An electronic image is formed with the help of pixels classified into these three categories.

If an image is not been able to put in any of these categories, then the category that is extremely near to the image is selected.

4. Parts of the Process

A vision system will consists of a small camera, a monitoring system (a computer) and the necessary hardware and software.

The whole process of identifying the image is classified into three main parts:

- **Image Processing**

Image processing is a process by which an image is formed for analysis and use at a later stage.

It uses various techniques such as image analysis and histogram of images to identify, simplify, modify or enhance an image.

- **Thresholding**

Threshold is a process in which each image is classified into various categories and then compared with the pixels stored in the database.

The pixels once compared are aligned to different levels to form an image.

- **Connectivity paths**

The connectivity path is a process by which a particular pixel is connected to a neighboring pixel if it falls in the same color and texture region.

It is the combination of all these three processes that a final electronic image is conceived and the required action is taken after analysis.

Check your Progress-6

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Types of Vision Systems

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

ii. Define Parts of the Process

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

iii. Define Image Processing

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

iv. Define Thresholding

…………………………………………………………………………

…………………………………………………………………………

14.7. Robotics information Category

1. Firefighting Robot for helping Firefighters



Firefighting robots has started to increase, many related improvements are taken so often to make it powerful. One of the latest inventions is the Segway-like Firefighting Robot developed by the Engineers at the University of California, San Diego. This firefighting robot helps in providing the happenings at a burning building.

2. Neato Unveils XV Signature Series with High Vacuum Power



Neato Robotics, a Silicon Valley company founded on an idea to create robots to free people from household chores. Their first robot was released in 2010, and today, the intelligent laser guidance system acts as the heart of all Neato robotic vacuums.

3. A new Self Driving RobotCar



A team led by Prof. Paul Newman at Oxford University, UK has demonstrated their Self Driving RobotCar on the private roads of Begbroke Science Park. It proves to be a great achievement as it can take over the driving once you accept the offer from the car's computer.

4. Autonomous Quadrotor MeCam streams video to your Smartphone



Always Innovating, a well-known company for its Touch Book and Smart Book has come up with a new flying video camera called as MeCam. It is a palm-sized autonomous quadrotor that has four spinning rotors to keep them aloft.

5. Keep your Swimming Pool Clean with iRobot Mirra 530



Mirra 530 is a pool cleaning robot which is capable of cleaning your pool surface and water thoroughly with its iAdapt Nautiq Responsive Cleaning Technology.

Humanoid Robots

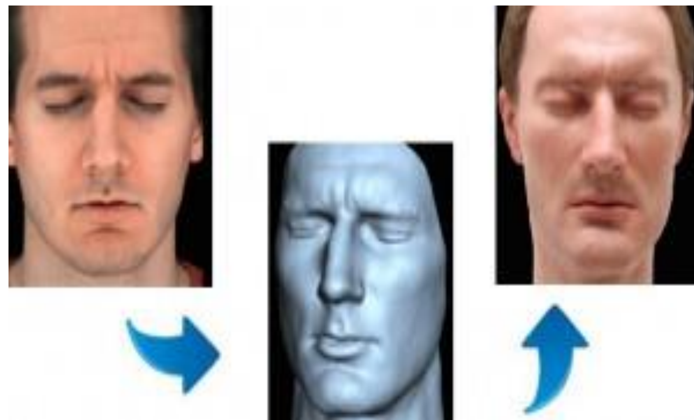
1. **Alissa – Russia’s first female android**



In this modern world, Android has been expected to be the future of robotics, and Neurobotics and Russia 2045 movement also expects the same to happen by 2045.

A little impact of it is shown from the development of Alissa, which is considered as the Russia’s first realistic female android head.

2. Disney invents a new face cloning method for robots



For long years, the robotics researchers have succeeded in developing humanoid robots, but failed in bringing the natural human faces and expressions. Therefore, Disney's researchers at Switzerland have come out with a 'face cloning' technique, which provides the most realistic facial expressions to the animatronic robots. This technique uses 3D Motion Capture Technology for scanning purpose.

3. fMRI allows robot to read human thoughts remotely



The fMRI, known as Functional Magnetic Resonance Imaging is a machine that can perform unbelievable things like recording videos of your dreams, enlightening innovative skills during sleeping, etc. Now, Israeli researchers have used it in controlling a robot remotely through human thoughts.

4. Robo-Glove reduces astronauts and autoworkers repetitive stress injuries



Robo-Glove is a robotic wearable device developed by NASA and General Motors (GM) for minimizing the repetitive stress injuries of astronauts and autoworkers. Generally, when a person holds a tool for longer time, he may feel fatigue in his hand muscles.

Medical Robots

1. **Raven II surgical robots**



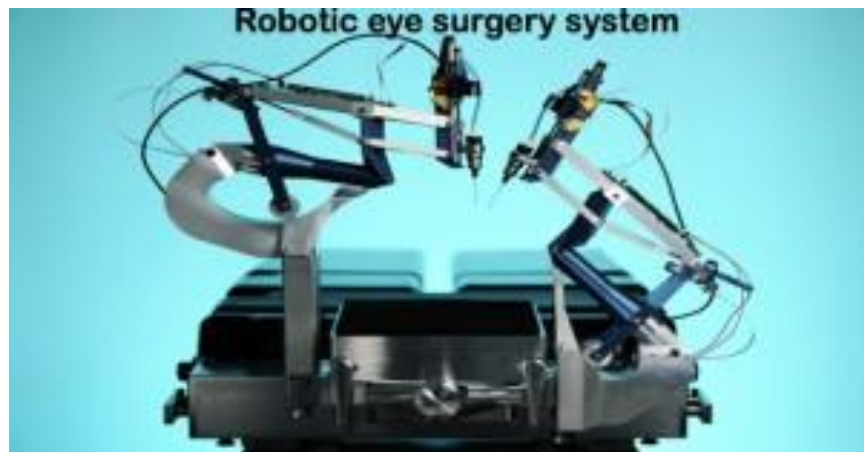
Raven II is the name given to the surgery robot, which was developed by the researchers of University of Washington and the University of California, Santa Cruz.

2. A robotic system helps to perform brain surgery



A neurosurgeon performing a keyhole neurosurgery makes a burr hole on the patient's head for accessing the brain. Several conditions like hydrocephalus, Tourette syndrome, tumors, and epilepsy can be cleared by this process.

3. Robotic eye surgery system



Thijs Meenink, a researcher and Ph.D. student of Netherlands' Eindhoven University of Technology has invented 'Robotic Eye Surgery System' for performing eye operations. This system is somewhat similar to the da Vinci robotic surgery system.

Military Robots

1. **DARPA LS3 Robot Moves to New Heights with New Additional Features**



DARPA has recently revealed a video of Legged Squad Support System (LS3) with new exciting features. The main purpose of LS3 is to carry a high payload from one place to another in a battle field.

2. **iRobot 710 Warrior is ready for action**



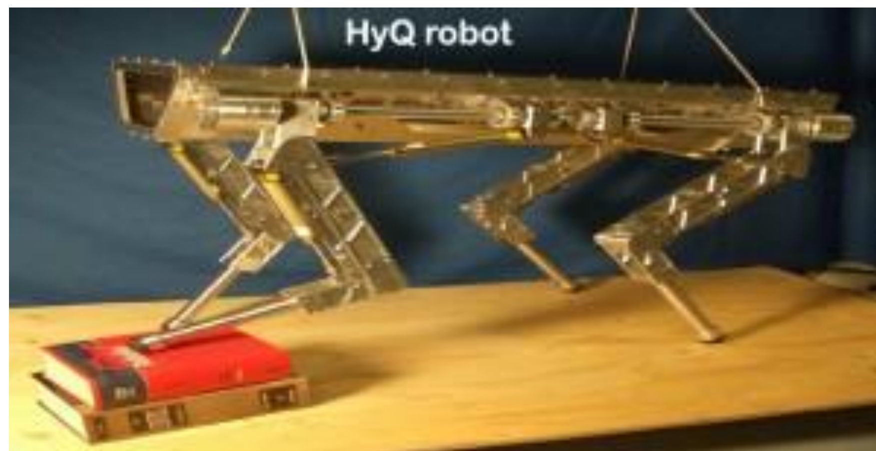
iRobot is a company which is well-known for its Roomba robotic vacuum. It has developed several useful robots like 110 FIRSTLOOK, 210 NEGOTIATOR, 510 PACKBOT, and so on. In this line up, the company has now included the updated version of Warrior 700 robot called as 710 Warrior.

3. Novatiq designs SCORP throwable robot



Novatiq, a Swiss company is ready to rock the robotics field with its first SCORP throwable robot. It is a rough, lightweight, and small Micro Unmanned Ground Vehicle (MUGV) especially made for scouting and surveillance applications.

4. HyQ – Hydraulically actuated quadruped robot



HyQ is a hydraulically actuated quadruped robot, which can run, jump, and as well as travel in the uneven terrains. It was developed by Professor Darwin Caldwell and his team members in the Department of Advanced Robotics at IIT.

5. Boston Dynamics unveils AlphaDog military robot



Boston Dynamics has shown the capabilities of their new AlphaDog quadruped robot in the IROS event last month at San Francisco. It was first known as the LS3 (Legged Squad Support System) robot. This robot is developed with the help of DARPA and US Marine Corps funds.

6. Bomb disposal robot



In this modern world, the significance of security force robots has been increased to a larger extent. Among them, the bomb disposal robot seems to be the most important one to be incorporated in the military, bomb squad, and other security departments for saving many valuable lives.

14.8 Machine Vision Category

1. Industrial robot draws human image autonomously



If you generally think of industrial robots, you may suddenly get its operating capabilities like machine loading & unloading, welding, and other industrial works on your mind.

2. Training methods for an industrial robot vision system

During training period, various objects are made familiar to the vision system of the industrial robots. The extracted feature values of these known objects are stored in the vision system, and then compared with the feature values of unknown objects.

3. Robotic applications of a machine vision system

A machine vision system is employed in a robot for recognizing the objects. It is commonly used to perform the inspection functions in which the industrial robots are not involved. It is usually mounted in a high speed production line for accepting or rejecting the work parts.

4. Automated inspection

Inspection is a quality control process that is concerned with the checking or testing of work parts against the certain conditions described by the design engineer. Inspection is performed in both incoming raw work parts and as well as finished work parts.

5. Machine Vision System

Machine vision system is a sensor used in the robots for viewing and recognizing an object with the help of a computer. It is mostly used in the industrial robots for inspection purposes. This system is also known as artificial vision or computer vision.

Robotic applications of a machine vision system

A machine vision system is employed in a robot for recognizing the objects. It is commonly used to perform the inspection functions in which the industrial robots are not involved.

It is usually mounted in a high speed production line for accepting or rejecting the work parts. The rejected work parts will be removed by other mechanical apparatuses that are in contact with the machine vision system.

A machine vision system can be incorporated with an industrial robot for performing the following three important tasks such as:

- Inspection
- Identification
- Visual servoing and navigation

1. **Inspection**

The industrial robots are only used to support the machine vision system when it performs the inspection tasks.

During this process, it checks for accurate surface finish, exact dimension, errors in labeling, presence of holes in the work parts, and other factors.

The machine vision system carries out this inspection processes automatically with less time and errors.

In addition, the human workers can also perform these operations manually, but there is a high possibility of error occurrence and increased operation time.

2. Identification

In this process, the machine vision system performs recognizing and categorizing of work parts instead of inspecting it.

It also helps in determining the work part’s position and orientation. Some of the operations accomplished by a machine vision system in the identification process are work part palletizing and depalletizing, object sorting, and gripping the parts oriented from a conveyor.

A robot is used in these tasks to take successive action and decision.

3. Visual servoing and navigation

In this application, a machine vision system controls the actions of a robot according to the visual input.

For example: In robot visual servoing process, a machine vision system directs the path of robot’s end effector to a work part in the work cell.

Some applications of this category consist of positioning of work parts, seam tracking, bin picking, and retrieving and re-orienting the work parts that are moving along a conveyor.

With the help of visual data, the navigational control can be used in collision protection and automatic path planning of a robot.

Check your Progress-7

Note: a. Write your answer in the space given below.

b. Compare your answer with those given at the end of the unit.

i. Define Inspection

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

ii. Define Identification

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

iii. Define Visual servoing and navigation

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

14.9 Unit – End Exercises

1. List out the Robotic applications of machine vision
2. Define Cell decomposition method
3. Define Sensors
4. Define Passive sensors
5. Define Active sensors
6. Define Inspection
7. Define Identification
8. Types of Vision Systems
9. Image Processing
10. Thresholding
11. Connectivity paths

14.10 Answer to Check your Progress

1. List out the applications of robot domain.

    a. Industry and Agriculture
    b. Transportation
    c. Hazardous environments
    d. Exploration
    e. Health care
    f. Personal Services
    g. Entertainment
    h. Human augmentation

2. The path planning are uses cell decomposition. It decomposes the free space into a finite number of contiguous regions, called cells.

3. Sensors are the perceptual interface between robots and their environments.

4. Passive sensors: such as cameras, true observers of the environment.

5. Active sensors: such as sonar, send energy into the environment.

6. The industrial robots are only used to support the machine vision system when it performs the inspection tasks. During this process, it checks for accurate surface finish, exact dimension, errors in labeling, presence of holes in the work parts, and other factors. The machine vision system carries out this inspection processes automatically with less time and errors.

7. It also helps in determining the work part's position and orientation. Some of the operations accomplished by a machine vision system in the identification process are work part palletizing and depalletizing, object sorting, and gripping the parts oriented from a conveyor. A robot is used in these tasks to take successive action and decision.

8. A robot's vision system is classified into three main types on the basis of the color of the objects. They are: Binary image, which consist of black and white images, Gray colored images, and Colored images with the base of red, green or blue.

9. Image processing is a process by which an image is formed for analysis and use at a later stage.

It uses various techniques such as image analysis and histogram of images to identify, simplify, modify or enhance an image.

10. Threshold is a process in which each image is classified into various categories and then compared with the pixels stored in the database.

The pixels once compared are aligned to different levels to form an image.

11. The connectivity path is a process by which a particular pixel is connected to a neighboring pixel if it falls in the same color and texture region.

It is the combination of all these three processes that a final electronic image is conceived and the required action is taken after analysis.

14.11 Suggested Readings

1. E. R. Davies (2004), Morgan Kaufmann, Machine Vision 3rd Edition Theory, Algorithms, Practicalities, UK.

2. Ramesh Jain, Rangachar Kasturi, Brian G. Schunck (1995), Published by McGraw-Hill, Inc., MACHINE VISION, ISBN 0-07-032018-7.

3. Stuart Russell and Peter Norvig (2003), Artificial Intelligence A Modern Approach, Second Edition Prentice Hall Series.

4. Mark S. Nixon, Alberto S. Aguado (2008), Feature Extraction and Image Processing, Academic Press is an imprint of Elsevier, Second edition, UK.

5. https://www.brighthubengineering.com/robotics/54373-vision-system-in-robots/

6. http://www.roboticsbible.com/category/industrial-robotics/ind-robo-vision-sys

DISTANCE EDUCATION – CBCS-(2018-19 Academic Year Onwards)

Question Paper Pattern (ESE) - Theory

(UG / PG / P. G. Diploma Programmes)

Time: 3 Hours Maximum: 75 Marks

Part-A (10 x 2 = 20 Marks)

Answer all questions

1. Define Artificial Intelligence.
2. List out the main goals of Artificial Intelligence.
3. Define Inference Rules.
4. Define Knowledge Engineering.
5. What is Expert System?
6. What is Expert System Shell?
7. Define Means End Analysis.
8. What is meant by Modeling?
9. Define Machine Vision.
10. Define Lighting.

Part-B (5 x5 = 25 Marks)

Answer all questions choosing either (a) or (b)

11. a. Explain the types of application in Artificial Intelligence?

Or

b. Discuss the Goal-based agents with diagram in detail.

12. a. Explain Probabilistic Reasoning.

Or

b. Explain about Bayesian Network with neat Diagram.

13. a. Explain the Expert system Components.

Or

b. Discuss the Characteristic features of Expert systems.

14. a. Explain Breadth-first Search.

Or

b. Discuss about task planning.

15. a. Explain the various Functions in a vision system in detail.

Or

b. Discuss about the types of Feature extraction methods in vision system.

Part - C (3 x 10 = 30Marks)

(Answer any 3 out of 5 questions)

16. Discuss the various types of approaches in Artificial Intelligence with diagram.
17. Briefly discuss about Pattern Recognition with example.
18. Explain the reasoning and knowledge acquisition.
19. Elaborate Graph planning**.**
20. Discuss the Object representations with various types.